Project Report

On

Video Based Mobility Monitoring Of Elderly People In Smart Home

for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Rohit Prajapat - 2020BITE052

Rahul Singh - 2020BITE053

Under the supervision of

Dr. Prabal Verma

Department of Information Technology

National Institute of Technology Srinagar

June 2024

CERTIFICATE

This is to certify that the project titled **Video Based Mobility Monitoring Of Elderly People In Smart Homes** has been completed by **Rohit Prajapat (2020BITE052)** and **Rahul Singh (2020BITE053)** under my supervision in fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Information Technology**. It is also certified that the project has not been submitted or produced for the award of any other degree.

Dr. Prabal Verma

Supervisor

Dept. of Information Technology

NIT Srinagar

STUDENTS' DECLARATION

We, hereby declare that the work, which is being presented in the project entitled **Video Based Mobility Monitoring Of Elderly People In Smart Home** in fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology**, in the session 2024, is an authentic record of our own work carried out under the supervision of **Dr. Prabal Verma**, Department of Information Technology, National Institute of Technology, Srinagar. The matter embodied in this project has not been submitted by us for the award of any other degree.

Dated: _____

(i) Name: Rohit Prajapat
Signature: _____

(ii) Name: Rahul Singh
Signature: _____

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our department head **Dr. Janib ul Bashir** for encouraging and allowing us to present the project on the topic **Video Based Mobility Monitoring Of People In Smart Home** at our department premises for the fulfillment of the requirements leading to the award of B.Tech degree.

We deeply express our sincere thanks to our project guide, **Dr. Prabal Verma**, for her valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of this project. We take this opportunity to express gratitude to all of the department faculty members for their help and support.We also place on record, our sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Rohit Prajapat (2020BITE052)

Rahul Singh (2020BITE053)

Contents

List of Tables

List of Figures

Abbreviations

CV	Computer Vision
IoT	Internet of Things
AI	Artificial Intelligence
CV	Computer Vision
ML	Machine Learning
CNN	Convolutional Neural Network

Chapter 1

Introduction

Over the years, analysis of human activity in video has experienced significant improvements due to its wide applications; these include areas like surveillance, sports, human-machine interaction, rehabilitation, health monitoring and robotics. In healthcare for instance, analyzing human movements remotely has enabled diagnosis at a distance as well as surveillance of vulnerable patients and detection of anomalous events. Numerous products and services have been developed to support Ambient Assisted Living (AAL), which aims to promote healthy, active, and happy aging. As the global population rapidly ages, with the number of older individuals expected to double over the next three decades, there is an urgent need for effective solutions to monitor and ensure their well-being.

The growing prevalence of neurodegenerative diseases among the elderly profoundly impacts their quality of life and ability to age healthily. Consequently, elderly individuals require periodic monitoring to assess their mobility and detect potential health issues early. However, many elderly individuals are reluctant or unable to visit health clinics regularly due to disabilities or logistical limitations, such as residing in remote areas. This often leads to infrequent assessments, delayed interventions, and increased healthcare costs. In this context, the ability to analyze and control motion and cognitive abilities is crucial for improving the social and clinical living conditions of the elderly. Studies have demonstrated a strong link between cognitive impairment and motion dysfunction, including deficits in gait and balance.Studies have demonstrated a strong link between cognitive impairment and motion dysfunction, including deficits in gait and balance. Therefore, video analysis of human movements can significantly aid in assessing individuals' motion abilities,

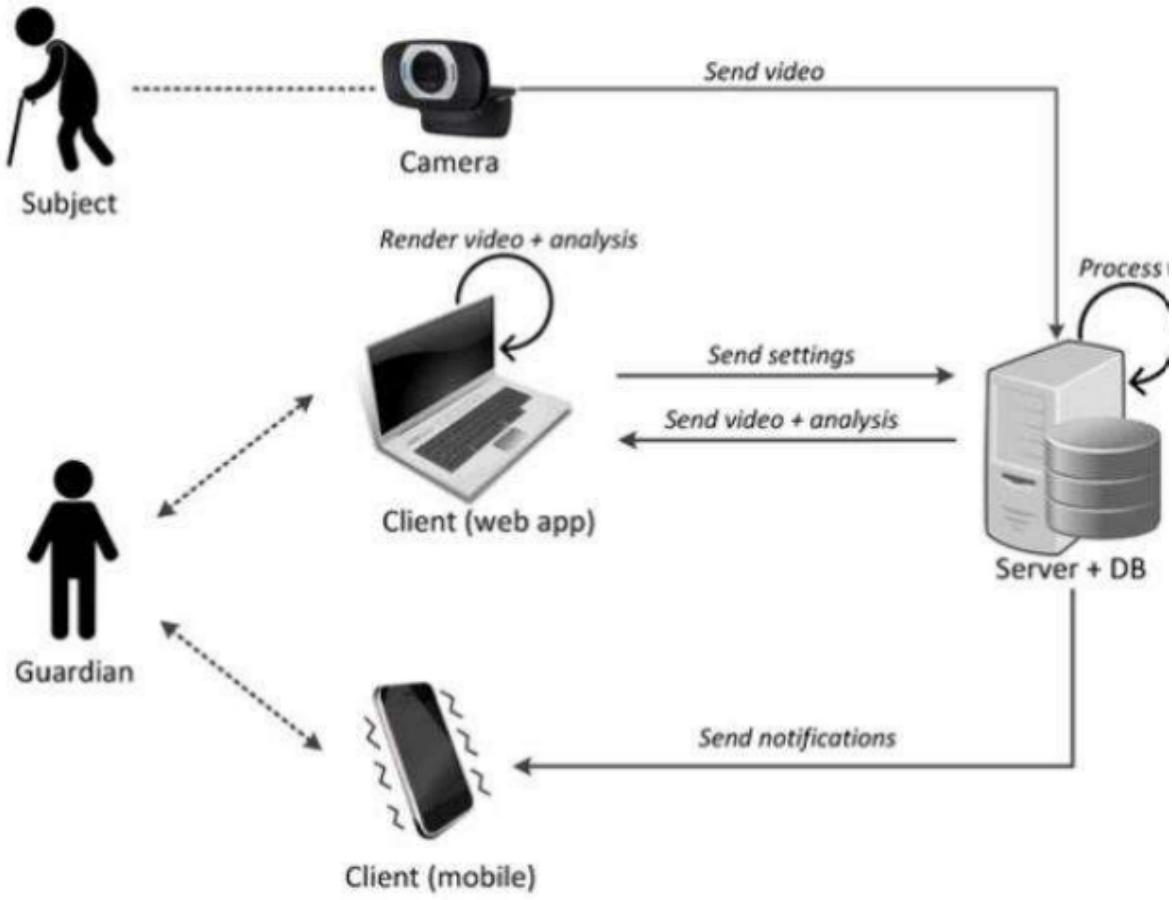


Figure 1.1: Working of project

providing objective evaluations and supporting remote diagnosis.

The proposed project suggests a system based on sight that employs the YOLOv8 model in order to detect falls and track movements of old people. The elderly people's video information at smart home settings is continuously being analyzed by cheap commercial RGB cameras and the results presented herein below. The video data are processed to detect falls with high accuracy, and upon detection, immediate alerts are sent to caregivers and healthcare providers via email. This prompt notification system ensures timely medical intervention, potentially reducing the severity of injuries resulting from falls.

The data is processed here to ensure correct fall detection, which sends immediate messages by email to guardians or medical personnel. The preliminary stage of this project involved the determination of falls from real-time video streams using OpenCV with respect to body angles. This nevertheless merely provided an overview of the fall detection in general. Therefore, this project

also incorporated the YOLO (You Only Look Once) family models that are renowned for their high accuracy and efficiency in object detection tasks as well as their speed. Consequentially, YOLOv5 was trained on a comprehensive dataset on fall detection, resulting in a remarkable improvement at 97% accuracy level.

To enhance its functionality, it had been further developed via training YOLOv8 on the same set and fine-tuning various parameters for obtaining an impressive 99

Other parts involve creating an e-mail notification system that will alert them if there is a fall. Once again, the SMTP library comes into play in sending emails with threading so that notification does not meddle with real time detection ability that guarantees immediate alerts sent out either to caregivers or loved ones

Additionally, the system extends its capabilities beyond fall detection to classify various mobility states such as standing, sitting, walking, and lying down. This comprehensive monitoring approach provides valuable insights into the daily activities and health status of elderly individuals, aiding in the early detection of mobility-related issues. By leveraging advanced deep neural network architectures, the system enhances its accuracy and reliability, emulating the decision-making processes of expert physiotherapists to provide objective and quantitative evaluations. This innovative approach aligns with the goals of Ambient Assisted Living, aiming to improve the safety, independence, and overall well-being of elderly individuals.

With this project we expect to come up with an effective and scalable solution for monitoring elderly mobility and detecting falls at low cost hence improving their life quality while lessening burden on healthcare sector. ultimately enhancing the quality of life for elderly individuals and reducing the burden on healthcare systems. By integrating advanced machine learning models with real-time monitoring and efficient notification mechanisms, this project offers a comprehensive solution that can be easily integrated into existing home security systems or used as a standalone monitoring tool.

1.1 Motivation

The critical need behind this project is to have an efficient fall detection system for old people in real-time. Elderly people are prone to falls that are always resulting in severe injuries, long-

term disabilities and increased mortality levels. Therefore, early identification of these falling seniors and prompt intervention can significantly improve their lives. Early detection and timely intervention are crucial in mitigating these adverse effects. However, many elderly individuals are reluctant or unable to visit health clinics regularly due to disabilities or logistical limitations, such as residing in remote areas. This often leads to infrequent assessments, delayed interventions, and increased healthcare costs.

Therefore, early identification of these falling seniors and prompt intervention can significantly improve their lives. In this regard, the ability to analyze and control motion/motion patterns including cognition is necessary to enhance the societal and medical living conditions of the aged population. Previous studies have shown a strong relationship between cognitive impairment and movement dysfunction such as gait disoders or balance deficits. Thus, video analysis of the human movements can be highly useful in assessing persons' mobility performance by giving objective evaluations and supporting remote diagnosis.

Our project proposes a vision-based system that leverages the YOLOv8 model to detect falls and monitor the mobility of elderly individuals in real-time. By using low-cost commercial RGB cameras, the system continuously analyzes video data from elderly residents in smart home environments. The video data are processed to detect falls with high accuracy, and upon detection, immediate alerts are sent to caregivers and healthcare providers via email. This prompt notification system ensures timely medical intervention, potentially reducing the severity of injuries resulting from falls.

Moreover, apart from fall detection, the system has been equipped with capacities allowing it to classify various mobility states like sitting or standing or walking or lying down. Through this comprehensive monitoring approach, one can get valuable information about daily activities as well as health condition of elderly individuals thus helping them find out any mobility related problems at early stages. By using advanced deep neural network architectures for improved accuracy and reliability, akin to expert physiotherapists' decisions making processes provided by emulated decision networks of their own making physiotherapists to provide objective and quantitative evaluations.

1.2 Problem Statement & Objectives

Problem Statement

In particular, falls pose a grave challenge in the health and safety of older people due to ageing population worldwide. However, the current methods of monitoring them, which are normally through clinical visits or some stationary alert system, cannot provide the necessary interventions and help in time. An advanced, real-time system that accurately detects falls and tracks mobility patterns among elderly individuals within their home environments is urgently needed.

Objective

The objective of this project is to develop an innovative vision-based system for real-time fall detection and mobility monitoring of elderly individuals in smart home environments. This system aims to achieve the following objectives:

1. **High Precision Fall Detection**:For clinical applications with precision levels suitable for video streams in real-time YOLOv8 architecture is leveraged by implementing and optimizing a deep learning model to detect falls correctly.Rather than this, the resulting system distinguishes between falls and other activities of daily living to keep false alarms at bay. The system will distinguish falls from other daily activities to minimize false alarms.

2. **Real-time Alert System**: Develop a robust alert mechanism that instantly notifies caregivers and healthcare providers via email alerts and mobile notifications upon detecting a fall or mobility issue. This system will ensure timely intervention and support, potentially reducing the severity of injuries resulting from falls.

3. **Cost-effective and Expandable Approach**: In addition to the sophisticated deep learning algorithms, researchers can utilize inexpensive RGB cameras in order to have scalable solutions easily deployable on several smart homes that are even found in the rural or undeserved areas. This approach will make it possible for a designed solution to work efficiently on the average household hardware.

4. **Extensive Mobility Assessment**: Go beyond mere fall detection and include monitoring as well as classification of other mobility states such as standing, sitting, walking and lying down in this system. Such a holistic evaluation will provide insight into what caregivers and health workers need to know about their aging patients' daily routines and physical state.

5. **Autonomous Living Support**: In order not to invade old people's privacy , but still be able to offer care only when needed, elderly individuals must be allowed to maintain their independence and privacy though unobtrusive monitoring combined with help when necessary . While respecting privacy concerns ,the system ought not only be prompt in case of emergencies but also must guarantee the same.

6. **Internet-of-Things (IoT) Integration**: The accuracy of the system could be enhanced and its functionalities extended by exploring possibilities of integrating it with IoT devices such as wearables sensors or smart home kits. Thus, more holistic methods of tracking aged peop health and well-being.

1.3 Organization

The rest of the report is organized as follows. In Chapter 2, we survey the relevant literature. Chapter 3 discusses our system design and architecture, Chapter 4 cover methodology and their integration into the existing system. Chapter 5 represents all the implementation-related details,and chapter 6 cover experiment and result, chapter 7 conclusion and future work.

Chapter 2

Literature Survey

In this chapter, we review the literature on video-based mobility monitoring systems for elderly people in smart homes to understand their operation, methodologies, and the challenges they face.

Video Based Mobility Monitoring of Elderly People Using Deep Learning Models [1]: This paper demonstrates the possibility of assessing elderly people's motion skills through a non-invasive, cheap visual camera-driven automated system that extracts features and builds models to replicate the decision-making process of physiotherapists Results show improved performance due to convolutional blocks, despite the limited dataset size. The study highlights the need for larger, more consistent datasets and suggests future enhancements using RGB-D cameras. The system supports timely detection of mobility anomalies, potentially improving telehealthcare through mobile apps for remote monitoring and assessment.

FallDetectNet: A Computer Vision Platform for Fall Detection [2]:In this paper,The authors describe FallDetectNet, which is a computer vision framework for fall detection. They tried transfer learning models with/without data augmentation, finding that pre-training significantly improved the model's generalization over small UR Fall Detection dataset. Conversely, the baseline CNN without data augmentation performed badly due to inadequate training data. The study also plans to create an alternative loss function designed to punish false negatives in order to improve precision at the expense of recall and will further investigate other datasets in order to enhance model robustness.

Fall Detection for The Elderly using YOLOv4 and LSTM [3]:The aim of this study is to address falls among the old age within homes using advanced techniques i.e., YOLOv4 for posture detection and LSTM for sequence analysis of postures. YOLOv4 was selected because it has good object detection efficiency that is crucial for accurate identification human postures related to fall detection. This can be complemented by LSTM.Enhancing the model's ability to recognize patterns indicative of falls.

Fall Detection For Elderly People Using Machine Learning [4]:The paper is about a fall detection system for elderly that uses wearable sensors and the application of machine learning algorithms in analyzing activities of daily living. Machine learning allows us to have fewer false alarms based on some inherited gait patterns. Decision tree algorithm gives better results compared to SVM as it accurately separates attributes into each class. The models' performance is confirmed by evaluation metrics such as sensitivity, specificity, accuracy and confusion matrices. Future enhancements could focus on expanding the training dataset and identifying optimal features to further improve accuracy.

Latest Research Trends in Fall Detection and Prevention Using Machine Learning: A Systematic Review [5]:This study seeks to criticize existing literature on wearables and statistical/threshold based approaches with high false positives. It highlights trends in current fall detection and prevention systems which focus on Machine Learning (ML) algorithms, recent studies, analysis of datasets, age groups, ML techniques, sensors and deployment locations. Highlighting ML's efficacy in reducing false alarms and improving accuracy, the paper discusses future directions for enhancing these systems.

Name	Description	Year
Video Based Mobility Monitoring of Elderly People Using Deep Learning Models	A system for automatically assessing elderly mobility skills through visual cameras and BiLSTM-based classification will be shown. Points include improved performance with bigger dataset being required.	2023
FallDetectNet: A Computer Vision Platform for Fall Detection	Methods used in improving precision-recall disparities in fall detection using transfer learning and data augmentation.	2021
Fall Detection for The Elderly using YOLOv4 and LSTM	Focuses on YOLOv4 for posture detection and LSTM for sequence analysis to improve elderly fall detection.	2022
Fall Detection For Elderly People Using Machine Learning	Fall detector based on a wearable sensor using decision tree algorithms for minimizing the number of false alarms and enhancing accuracy.	2020
Latest Research Trends in Fall Detection and Prevention Using Machine Learning: A Systematic Review	This should critique previous works emphasizing ML algorithms, and discusses trends in improving fall detection systems for the elderly.	2021

Table 2.1: Literature Review on Fall Detection Systems

Chapter 3

System Design and Architecture

In this chapter, we go over the architecture and security methods used in our project, along with all the necessary details which are needed to understand the working are listed here

YOLO

YOLO model came up in a document named "You only look once, Unified, Real-time object detection" written by Joseph Redmon. The algorithm accepts image as input and produces predicted bounding boxes and class labels for one grid cell using a single neural network. This model divides the given images into a grid of many cells with each cell estimating an approximated dimension of the bounding box consisting x, y coordinates , width, height along with confidence score. Detection process includes non-max suppression. Even though it is fast in detection, localizing smaller objects is not precise.

3.0.1 Unified Detection System

Each of the input images in YOLO is divided into an S × S network of cells. A cell within the network that

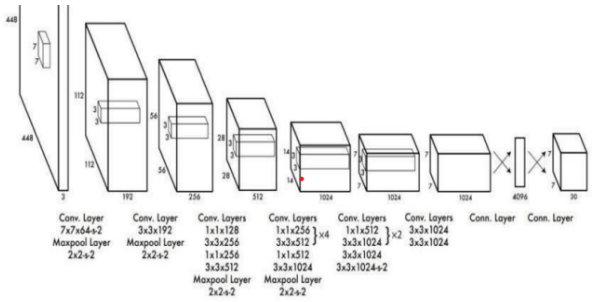
contains an object center will be responsible for detecting it. For each cell in YOLO (e.g., i-th), we calculate p-c(i) which gives us the probability that there is anobject in thisbox. The accuracy of the item class prediction and the possibilitythat there is an object in the box are reflected in these

confidence scores. The confidence

Intersectionover Union(IoU) metric for actual and prediction of bounding boxes. If the confidence value is zero It states that the grid cell doesn't contain any object. At a threshold confidence value equals mAP is calculated. When confidence is lessthan the threshold value then the bounding boxes are not considered. For example every object consists a bbx,bby,w,h and conf attributes where bbx=bby=0 represent grid origin such that w = h = 1 indicates without any size constraint on bbox's height or width.(bbx,bby) coordinates denote position at center of bbox relative to restrictions put forth by its corresponding gridcell.The w and h are predicted relative to the whole image. Each grid cell predicts theclassprobabilities (Pr(Class)) of the objects it contains

YOLOv5

YOLOv5 is one of the models in You Only Look Once (YOLO) family computer vision models. Object detection is a main use case for YOLOv5. There are four main versions of YOLOv5: small (s), medium (m), large (l), and extra-large (x). The smaller it is, the more accurate it becomes. Each variant has its own training time.



The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Figure 3.1: Tranining By YOLOv5

In the chart above, you can see that all variants of YOLOv5 train faster than EfficientDet. The most accurate YOLOv5 model, YOLOv5x, can process images multiple times faster with a similar degree of accuracy than the EfficientDet D4 model.

3.0.2 An Overview of the YOLOv5 Architecture

Object detection, which YOLOv5 was designed for, involves creating features from input images. These features are then passed through a prediction system to draw boxes around objects and predict their classes

The YOLO network consists of three main pieces:

- **Backbone**: A convolutional neural network that aggregates and extracts image features at different granularities.
- **Neck**: A series of layers designed to mix and combine the image features extracted by the backbone. These layers prepare the features for the final prediction steps.
- **Head**: The head of the network consumes the features from the neck and performs the final steps of box and class predictions based on these features.

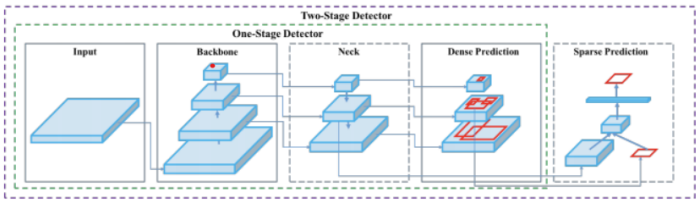


Figure 3.2: Object Detection Process

YOLOv8

YOLOv8 can be seen as an improved version of the YOLO series that further focuses on improving accuracy and performance. Here's an overview of YOLOv8:

1. **Architecture**:
    - YOLOv8 might use a more sophisticated backbone network, possibly integrating attention mechanisms or other advanced features to better capture context and spatial information.

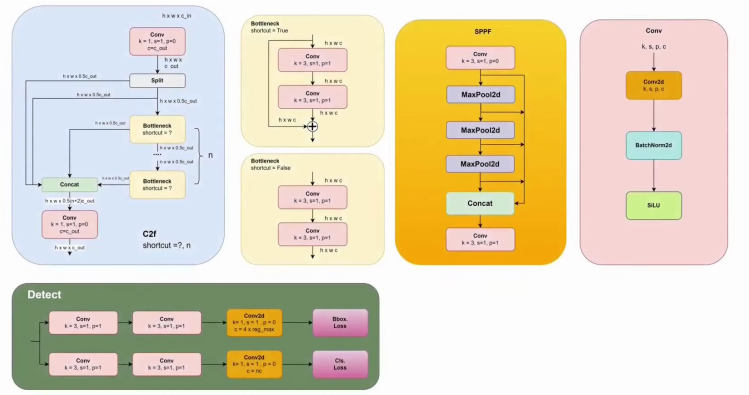


Figure 3.3: Architecture YOLOv8

2. **Loss Function**:

   - Unlike YOLOv5, which would combine localization and confidence losses, possibly with modifications specific to certain improvements in accuracy and robustness

3. **Training**:

   - Training YOLOv8 includes providing fall-specific annotated datasets so that it learns to detect falls accurately.it learns to detect falls accurately.

4. **Parameter Tuning**:

   - YOLOv8 To optimize performance, one may wish to fine-tune various hyperparameter like learning rates, batch sizes, augmentation strategies used during training phaseensuring.

Formulations

- **Bounding Box Prediction**: YOLO predicts bounding boxes using the following formula:

$$b_x = \sigma(t_x) + c_x, \quad b_y = \sigma(t_y) + c_y$$

  where b_x, b_y are the coordinates of the center of the bounding box, t_x, t_y are predicted offsets, and c_x, c_y are the coordinates of the grid cell.

- **Class Prediction**: YOLO predicts class probabilities using softmax activation:

$$Pr(Class_i|Object) = \frac{\exp(\text{score}_i)}{\sum_j \exp(\text{score}_j)}$$

where score_i is the confidence score for class i.

Key Features of YOLOv8

YOLOv8 has brought in some key features that set it apart from earlier versions:

- **Anchor-Free Architecture**:Instead of the traditional anchor-based detection YOLOv8 goes for an anchor-free approach. Such a shift makes training somewhat easier and allows the model to be compatible with various datasets.

- **Advanced Data Augmentation**:By using methods like MixUp and Mosaic, YOLOv8 toughens up the model and helps it work well in real-world applications. This strategy is meant to improve the accuracy of the model by providing diverse images during training.

- **Adaptive Training**: The learning rate can adjust itself dynamically using this feature; besides, it also helps balance out losses effectively during training leading to better performance optimization and higher detection accuracies.

- **Self-Attention Mechanism**: YOLOv8 introduces a self-attention mechanism that enables the model to understand how different features in an image are related or rely on each other. In particular, this is helpful for complex scenes where understanding context matters most.

- **Improved Backbone and Neck Architectures**: For accurate object detection, the model uses state-of-the-art architectures for feature extraction. These changes support efficient handling of diverse object detection tasks by the model.

- **Efficiency and Speed**: However, despite all these improvements, YOLOv8 still manages to strike a balance between accuracy and speed making it suitable for real-time object detection applications.

3.0.3 YOLOv8 vs. YOLOv5

After trying both models on Roboflow 100 we found that YOLOv8 got an mAP score of 80.2 compared to 73.5 mean score for YOLOv5 which means that YOLOv8 is much better at domain-specific tasks than Ultralytics' YOLOv5 predecessor. We compared YOLOv5 and YOLOv8 in this analysis.

3.1 OpenCV

OpenCV (Open Source Computer Vision Library) is a freely available computer vision library that aims to solve computer vision problems and it provides a comprehensive set of tools for tasks such as image processing, object detection, video analysis etc.

3.1.1 Introduction to MediaPipe

MediaPipe developed by Google Research is a framework that helps in building multimodal machine learning pipelines. This framework has a collection of calculators that are modular and they assist in performing tasks such as pose estimation, object detection and gesture recognition. The different components of the system are connected together to create pipes which can process multimedia streams on various platforms including desktops, mobiles as well as embedded devices.MediaPipe integrates with TensorFlow for machine learning tasks, making it suitable for applications like augmented reality (AR), virtual reality (VR), interactive media experiences, and more. MediaPipe offers pre-built modules to simplify the development of multimedia applications and real-time optimization strategies.

3.1.2 How does OpenCV work

OpenCV often uses Viola-Jones algorithm for object detection which was proposed by Paul Viola and Michael Jones.Object detection using Haar-like features proceeds as follows

The first step involves training a cascade classifier with a large dataset of positive and negative labeled images. This classifier learns to identify "HAAR features," which are rectangular features with regions of contrasting pixel intensities.

A HAAR feature calculates the difference between light and dark areas within its rectangle based on pixel intensities.These features are computed across locations and scales on an image.: Initially, there are several ways through which new objects can be detected

however one of the most commonly used method is sliding window approach where an image is scanned for each window position. At each window position, the classifier evaluates whether the features extracted match those learned during training. If a window contains all the features with sufficient matching scores, it is identified as containing the object of interest.

To speed up detection, a cascading technique is employed. If a window fails at any stage of the cascade (indicating the absence of the object), further processing for that window is halted, and the algorithm moves to the next window. This improves efficiency by quickly rejecting non-object regions.

Viola-Jones algorithm simplifies the calculations such that they are computationally feasible despite the fact that the digital images may contain thousands of pixels. Detection accuracy and speed are improved through various mathematical optimizations and techniques.

3.2 SMTP and Email Alerts

Simple Mail Transfer Protocol (SMTP) is a standard protocol for sending electronic mails over internet. For integrating email notifications and alerts into applications, SMTP can be used in programming. It works by connecting to an SMTP server like Gmail or Outlook, then using commands defined in it to send email messages. This includes specifying sender and recipient addresses, setting message content and headers (such as subject and date), and handling attachments if required. SMTP is vital for automating communication and notifying users or administrators of events, errors or updates within applications

3.2.1 Threads to Create Alerts

Threads in programming allow for concurrent execution of tasks within a single process. These are lightweight processes that share memory space and can run on their own. Threads are utilized in applications for carrying out background operations, asynchronous tasks, real-time alerts etc. For example, in Python, threads are managed using the ‘threading‘ module, where developers can

create and control threads to monitor conditions, such as system metrics or user events without blocking main program flow so as to trigger alerts or notifications.This enables applications to be responsive while doing multiple things at once.

3.3 Data Flow Diagrams

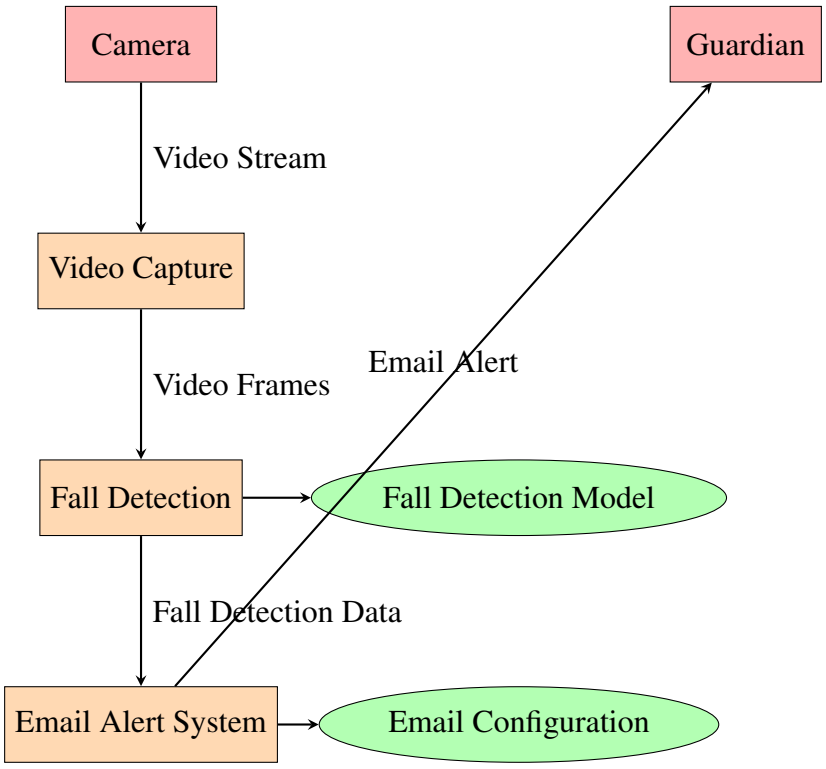


Figure 3.4: Data Flow Diagram for Real-Time Fall Detection System

3.3.1 Processes

- **Video Capture**: Captures real-time video feed from a camera.
- **Fall Detection**: Processes the video feed to detect falls using a trained YOLOv8 model.
- **Email Alert System**: Sends email notifications to guardians when a fall is detected.

3.3.2 Data Stores

- **Fall Detection Model**: The trained YOLOv8 model used for detecting falls.
- **Email Configuration**: Configuration details for sending email alerts.

3.3.3 External Entities

- **Camera**: The source of the real-time video stream.
- **Guardian**: The recipient of the email alerts.

3.3.4 Data Flows

- **Video Stream**: The continuous video feed from the camera to the video capture process.
- **Video Frames**: The frames processed by the fall detection system.
- **Fall Detection Data**: The result of the fall detection process.
- **Email Alert**: The notification sent to the guardian.

Chapter 4

Methodology

4.1 Data Collection and Preparation

To do this, I started by finding appropriate datasets from platforms like Kaggle and Roboflow. These datasets contained annotated images and videos depicting instances of falls among elderly individuals. Ensuring the quality and diversity of the data was crucial to training accurate models.

4.2 Annotation and Labeling

These features were meticulously annotated to identify key aspects of falls in each frame or image: for example, bounding boxes around individuals with particular emphasis on body orientation and positions that are indicative of a fall. This step ensured that the models could learn from accurately labeled data, improving detection accuracy during training.

4.3 Preprocessing for OpenCV and YOLO Models

The preprocessing steps carried out before training with OpenCV and later, YOLOv5 and YOLOv8 models were undertaken in order to normalize the input data format. These involved changing images or frames to one size, making them suitable for the model formats, and pixel value standardization. This is why such pre-processing steps were vital

in ensuring that the models could learn effectively from the input data without being hindered by

variations in image sizes or pixel intensities.

4.4 Data Splitting for Training and Validation

To evaluate accurate model performance,dataset divided into three set Training sets, Validation sets and Testing sets. Models were trained using the train set while validation set and testing sets. The training set was used to train the models, the validation set helped tune hyperparameters and prevent overfitting, while the testing set provided a final assessment of the models' performance on unseen data. This stratified splitting ensured that the models could generalize well to new instances of fall detection.

4.5 Handling Imbalanced Classes

Since falls are relatively rare compared to normal activities, classes might be imbalanced in the dataset. To address this imbalance oversampling was used during training or class weights implemented. This allowed models to be equally good at detecting falls as well as normal activities so that there would be reduced chances of false alarms or missed detections

4.6 Conclusion

By using advanced model such as YOLOv5 and YOLOv8 after preprocessing our data correctly , this project made realtime fall detection and show highly accurate. These are some of the preparatory steps that helped with subsequent stages of model development and integration into a live monitoring system.

Chapter 5

Implementation

In this chapter, we will use models that were used, their descriptions, training processes, and performance evaluations.The major implementation details listed in this chapter include hardware and software requirements where applicable , Fall detection algorithms, and realtime video process techniques for system integration and alert mechanisms.

5.1 Software Configuration

Different software tools used during the timeline of the project are described below:

Operating System: Windows 11.
Programming Language: Python 3.8
Deep Learning Frameworks: PyTorch 1.8
Computing Environments: Jupyter Notebook
Additional Libraries: cv2, mediapipe, numpy, smtplib, threading, asyncio, time.

5.2 Hardware Configuration

GPU: Intel Iris 20 GB and NVIDIA GPU
CPU: Intel i7 11th Gen @ 2.70GHz
RAM: 32 GB and 16 GB

Storage: 512 GB SSD and 2 TB SSD

5.3 Models Development

Each model contributed to improving the accuracy of fall detection systems and making them more efficient

5.3.1 Initial Model: OpenCV with Body Part Angle Detection

Description: Description: The starting point was using OpenCV to detect falls based on angles separating

different body parts. For this technique we had to identify some main points of human body then

compare their angles so as to confirm if it's a fall.

Performance: However, this model has some drawbacks which include low accuracy,

real-time detection issues, and reliability.

5.3.2 YOLOv5 Model

Description: As a result we switched to YOLOv5 ('You Only Look Once' version 5) object detection model known for its high accuracy. In essence, YOLOv5 is a new age deep learning model meant for any real time object detection mission.

Training: Specifically, our dataset used in training the model focused on detecting fall events.

Performance: Consequently, this model greatly improved the precision level of fall identification,

maintaining an error rate of about 3 percent only thus less likely to fail

5.3.3 YOLOv8 Model

Description: The latest version of the YOLO series, which is YOLOv8, was further developed to offer better improvements in architecture and training techniques for effective performance in

real-time detection tasks

Training: The YOLOv8 model also went through training on similar dataset about fall prevention but with some further parameter optimization for better performance on this specific task.

Performance: A solution that had an accuracy level of about 99% was given by the YOLOv8 model, thereby making it very reliable and highly accurate when it came to detecting falls in real-time.

5.4 System Integration and Real-Time Processing

this section explains how components were integrated and configured into a fall detection system. This included initializing real-time video capture that will be used as input source alongside configuring fall detection algorithm besides implementing email alert system while managing asynchronous operations.

5.4.1 Real-Time Video Capture

The video capture component of the system was implemented using OpenCV for continuous video stream processing:

- **Implementation:**
  The OpenCV's video capture functionality would process every single frame from videos fed by camera for subsequent fall detection processing
- **Details:**
  For video input handling, frame extraction and image preprocessing towards a fall detection model The OpenCV library was utilized. In order to achieve real-time video capture, the OpenCV library was employed.

5.4.2 Fall Detection

The fall detection algorithm was implemented using the YOLOv8 model, which was optimized for real-time operation:

- **Algorithm:**

  To detect falls happening in real-time, the YOLOv8 model was used. For object detection tasks, the latest version of YOLO series is YOLOv8 which has high accuracy and efficiency. For video input handling, frame extraction and image preprocessing towards a fall detection model.

- **Configuration:**

  Parameters of the model were adjusted for accurate fall detection; as part of this exercise, hyperparameter tuning, optimization of model architecture and thresholds for fall detection were done.

5.4.3 Email Alert System

The email alert system was designed to send notifications when a fall is detected:

- **Implementation:**

  The `smtplib` library was used to send email alerts. When a fall was detected, an email notification was sent to the guardian.

- **Threading:**

  Threading enabled implementation of an email-sending application that will not block real-time fall detection operations. This allowed the fall detection and email notification processes to run concurrently.

5.4.4 Asynchronous Operations

To manage real-time processing tasks and email alerts effectively:

- **Management:**

  Asynchronous operations were handled using the `asyncio` library along with `time` to manage timing and concurrent tasks.

- **Details:**

  `asyncio` was used for handling asynchronous tasks to ensure that the real-time processing of video frames and sending of email alerts occurred smoothly and efficiently.

5.5 Datasets Used

5.5.1 Fall Detection Computer Vision Project

I have taken datasets from Roboflow. It consists of images labeled for different states relelated to fall detection.

Classes: The dataset includes 3 classes: fall, no fall, and other.

Number of Samples:

- **Train Set:** 70% of the dataset, consisting of 6901 images
- **Validation Set:** 20% of the dataset, consisting of 1972 images
- **Test Set:** 10% of the dataset, consisting of 985 images

5.5.2 Fall Dataset

The dataset for this fall detection project is taken from Keggle. It consists of images labeled for different states related to fall detection.

Classes: The dataset includes 3 classes: fall, no fall, and other.

Number of Samples:

- **Train Set:** Consisting of 374 images
- **Validation Set:** Consisting of 111 images

Image Size: The images are typically of size 720x480 pixels.

‘

5.6 Mathematical Formulas

To ensure good performance of the fall detection models, we have used several mathematical metrics. The following formulas are used to quantify the effectiveness of the model.

5.6.1 Detection Accuracy

The accuracy of the model is calculated as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number of Samples}}$$

5.6.2 Precision, Recall, and F1 Score

To evaluate the performance of the fall detection model, we also use precision, recall, and the F1 score:

Precision:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

5.6.3 Intersection$_{O}ver_{U}nion(I0U)$

To evaluate the accuracy of object location in the our fall detection model, we use Intersection over Union (IoU):

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

5.6.4 Confusion Matrix Elements

A confusion matrix is used to evaluate the performance of the fall detection model:

- **True Positive (TP):**

  Number of correctly detected falls.

- **True Negative (TN):**

  Number of correctly identified non-falls.

- **False Positive (FP):**

  Number of falsely detected falls.

- **False Negative (FN):**

  Number of missed falls.

5.6.5 Mean$_{A}verage_{P}recision(mAP)$

For result of the fall detection model, we use (mAP):

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{n} \text{AP}_i$$

where AP_i is the Average Precision for the i-th class, and n is the number of classes.

5.7 Implementation using OpenCV

5.7.1 Importing Libraries and Setting Up Mediapipe

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

mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
```

5.7.2 Setting Up the Video Capture

```

cap = cv2.VideoCapture(0)
```

5.7.3 Function to Calculate Angles

```
1
2
3  def calculate_angle(a, b, c):
4      a = np.array(a) # First point
5      b = np.array(b) # Mid point
6      c = np.array(c) # End point
7
8      radians = np.arctan2(c[1] - b[1], c[0] - b[0]) - np.arctan2(a[1] - b[1], a
           [0] - b[0])
9      angle = np.abs(radians * 180.0 / np.pi)
10
11     if angle > 180.0:
12         angle = 360 - angle
13
14     return angle
```

5.7.4 Main Loop to Process Video Frames

```
1
2
3  with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5)
       as pose:
4      while cap.isOpened():
5          ret, frame = cap.read()
6          image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
7          image.flags.writeable = False
8
9          results = pose.process(image)
10         image.flags.writeable = True
11         image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
12
13         try:
14             landmarks = results.pose_landmarks.landmark
```

5.7.5 Extracting Key Points

```
1
2         shoulder_l = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,
              landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
3         hip_l = [landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x, landmarks[
              mp_pose.PoseLandmark.LEFT_HIP.value].y]
4         knee_l = [landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].x,
              landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].y]
5         # Other key points are extracted similarly...
```

5.7.6 Calculating Angles for Key Points

```
1
2
3         angle_elbow_l = calculate_angle(shoulder_l, elbow_l, wrist_l)
4         angle_elbow_r = calculate_angle(shoulder_r, elbow_r, wrist_r)
5         angle_shoulder_l = calculate_angle(elbow_l, shoulder_l, hip_l)
6         # Other angles are calculated similarly...
```

5.7.7 Visualizing the Key Points and Angles

```
1
2         cv2.putText(image, str(angle_elbow_l),
3                 tuple(np.multiply(elbow_l, [640, 480]).astype(int)),
4                 cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)
5         cv2.circle(image, (int(dot_LEFT_FOOT_INDEX_X), int(
              dot_LEFT_FOOT_INDEX_Y)), 5, (204,252,0), -1)
6         # Other visualizations for angles and points...
```

5.7.8 Calculate midpoints

```
1         # Calculate midpoints
2         def calculate_midpoint(point1, point2):
```

```
            return [(point1[0] + point2[0]) // 2, (point1[1] + point2[1]) //
                2]

        # Calculate midpoints for various points
        dot_LEFT_LEG_HIP
```

5.7.9 Fall Detection Logic

```

        fall = int(Point_of_action_X - dot_BODY_X )
        falling = abs(fall) > 50
        standing = abs(fall) < 50
        if falling:
            stage = "falling"
        if Point_of_action_X < 320 and Point_of_action_Y > 390 and standing
            and stage == 'falling':
            cv2.putText(image, 'fall', (320, 240), cv2.FONT_HERSHEY_SIMPLEX,
                2, (0, 0, 255), 2, cv2.LINE_AA)
            stage = "standing"
            counter_three += 1
        # Other fall detection conditions...

    cv2.imshow('Mediapipe Feed', image)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

5.8 Implementation using YOLOv8

5.8.1 Importing Libraries

```
import cv2
```

```
import torch
import torchvision
import matplotlib.pyplot as plt
import numpy as np
from ultralytics import YOLO
import smtplib
import threading
```

5.8.2 Checking GPU Availability

```
if torch.cuda.is_available():
    print("GPU is available.")
else:
    print("GPU is not available. Using CPU.")
device = 'cuda' # Use GPU if available
```

5.8.3 Loading and Training the YOLO v8 Model

```
model = YOLO('yolov8n.pt').to(device)
results = model.train(data='FallDetect.yaml', epochs=100, batch=3, workers=4,
    patience=0, lr0=0.001, lrf=0.0012, close_mosaic=0)
```

5.8.4 Loading the Trained Model

```
model = YOLO('runs/detect/train10/weights/best.pt')
```

5.8.5 Defining the Email Sending Function

```
def send_email(stage):
    try:
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login('laluyadavyadav717@gmail.com', 'ivws clno oaap jfwx')
```

```
        server.sendmail('laluyadavyadav717@gmail.com', 'rahulshiksingh@gmail.com
            ', stage)
        print('Mail sent')
    except Exception as e:
        print(f'Failed to send email: {e}')
    finally:
        server.quit()
```

5.8.6 Sending Email in a Separate Thread

```
def send_email_in_thread(stage):
    email_thread = threading.Thread(target=send_email, args=(stage,))
    email_thread.start()
```

5.8.7 Dictionary to Track Alerts Sent

```
Alert_sent = {}
```

5.8.8 Listing Available Cameras

```
def list_available_cameras(max_cameras=10):
    available_cameras = []
    for camera_id in range(max_cameras):
        cap = cv2.VideoCapture(camera_id)
        if cap.isOpened()):
            available_cameras.append(camera_id)
            cap.release()
    return available_cameras

# List all available cameras
cameras = list_available_cameras()
print(f"Available cameras: {cameras}")
```

Chapter 6

Experiments and Results

6.1 Experiments and Result for Fall Detection

6.1.1 Detection using OpenCV

initially provided a basic level of accuracy in localizing falls based on body part angles. However, this method showed limitations in terms of precision and reliability. The accuracy of fall detection using OpenCV was observed to be around 70%.

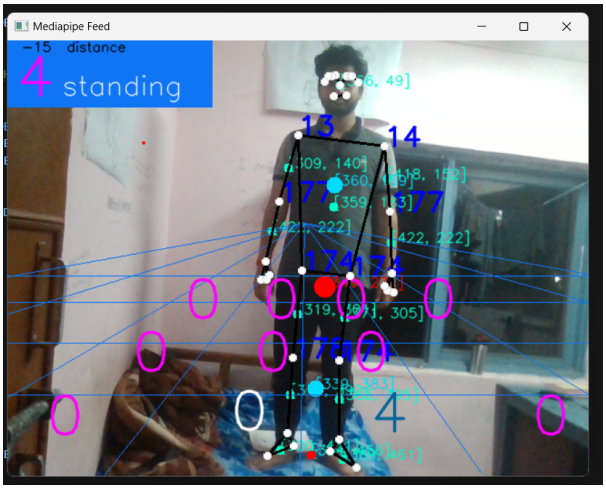


Figure 6.1: Standing Detected

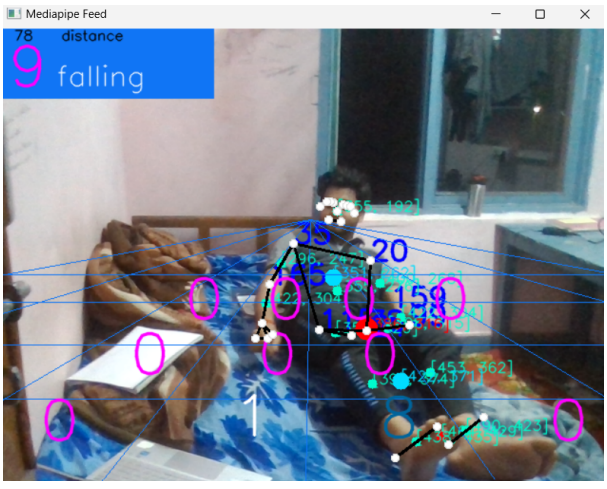


Figure 6.2: Falling Detected

While this method provided a basic level of fall detection and worked better than no detection at all, it had notable limitations.The strengths of the OpenCV model included its ability to detect

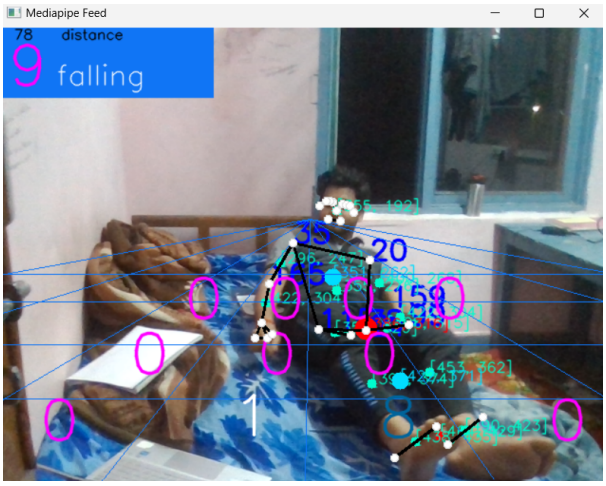


Figure 6.3: Works Poorly If Whole Body Parts Not Visible

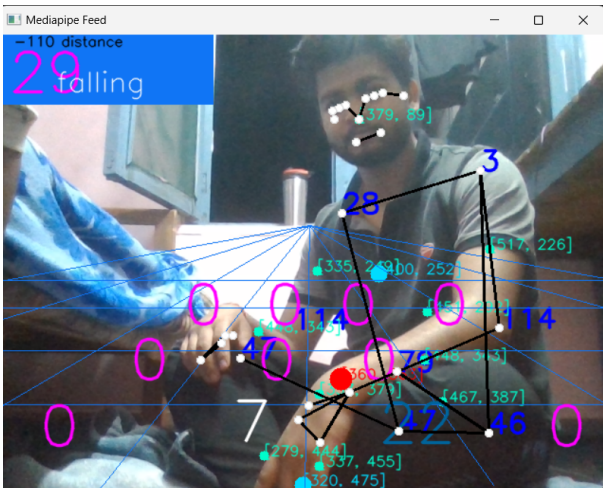


Figure 6.4: It shows Falling while Sitting

when people fall in seating position Significant body posture alterations that indicate a fall, even if the affected person is seat-ed, could be identified by the model.But it can't produce correct results due to close distance between a person and video,difficulty with frame and body angle.But it can't give proper results when a person too close to the video, finds it hard to find body angle and frames.

6.1.2 Detection using YOLOv5

YOLOv5 demonstrates locally accurate objects localization in terms of mean Average Precision metric which is a very good mAP when YOLOv5 shows 0.927 thresholds i.e 92.7% are acted and found as using YOLOv5 detection algorithm on average.Regardless that the improvement made by YOLOv5, there were still some other issues.The model was for high-resolution video feeds only and running these needed huge computation resources in real-time.Additionally, further fine-tuning was necessary to optimize its performance in specific settings with unique environmental conditions

here we used input images from dataset .

result of YOLOv5 output we get

Figure 6.5: Fall-Detection using YOLOv5

6.1.3 Detection using YOLOv8

Falls put senior citizens' health at great risk since they often result into serious injuries leading to medical complications.Accurate fall detection is only possible with timely intervention.This guarantees safety whilst giving serenity to both old people and their attendants.YOLOv8 application for detecting falls resulted in a mean Average Precision at Intersection over Union(IoU) threshold of 0.5 (mAP@50) equal to 0.99. The indication that the mAP is high tells us that this model is excellent.It makes it easier to locate where falls have occurred more precisely and differentiate between them and other activities better.

Figure 6.6: Fall-Detection using YOLOv8

YOLOv8 boasts higher accuracy due to enhancements in its architecture and training techniques. It achieved a mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 (mAP@50) of 0.99, compared to YOLOv5's 0.97. This improvement translates to more reliable fall detection, reducing both false positives and false negatives.

Here we can see As shown in Figure 6.6,that YOLO v8 provided way better accuracy than YOLOv5 in Figure 6.6 on same dataset

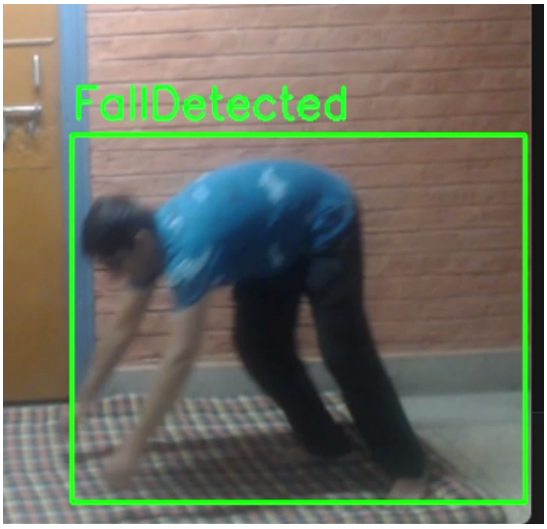


Figure 6.7: Fall Detected

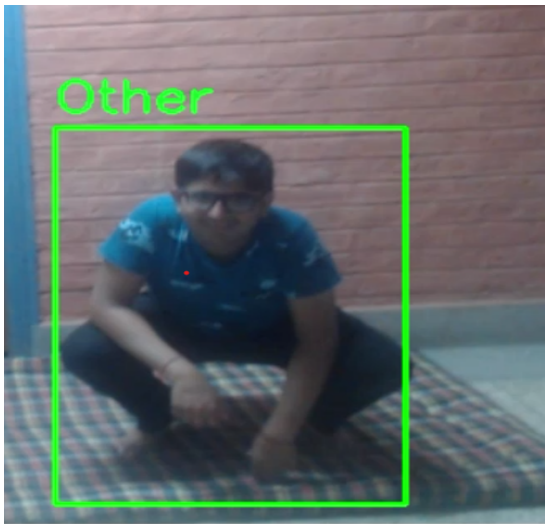


Figure 6.8: Other Condition

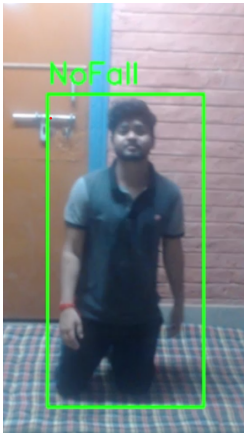


Figure 6.9: No-Fall Detected

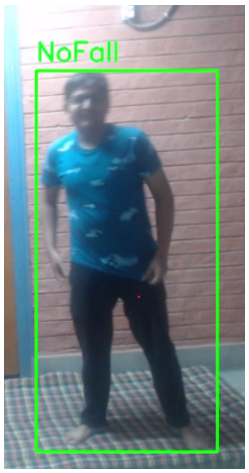


Figure 6.10: No-Fall Detected

6.2 Visulisation Result

6.2.1 Visulisation Result of YOLOv5

According to the author's opinion, YOLOv5 algorithm outperforms other commonly used algorithms in object detection. In particular, the YOLOv5 obtains an mAP-50 (Mean Average Precision at 50 percent Intersection over Union) of about 60, compared to the YOLOv3 mAP-50 of nearly 55. In terms of class-specific performance, YOLOv5 attains an AP (Average Precision) of 0.980 for the upward class and 0.969 for the downward class, resulting in a generated mAP of 0.975.

These results indicate that YOLOv5 excels in precision for both object orientations and shows significant improvements in detection accuracy when compared to other algorithms on the VOC dataset, highlighting its robustness and effectiveness in object detection tasks.

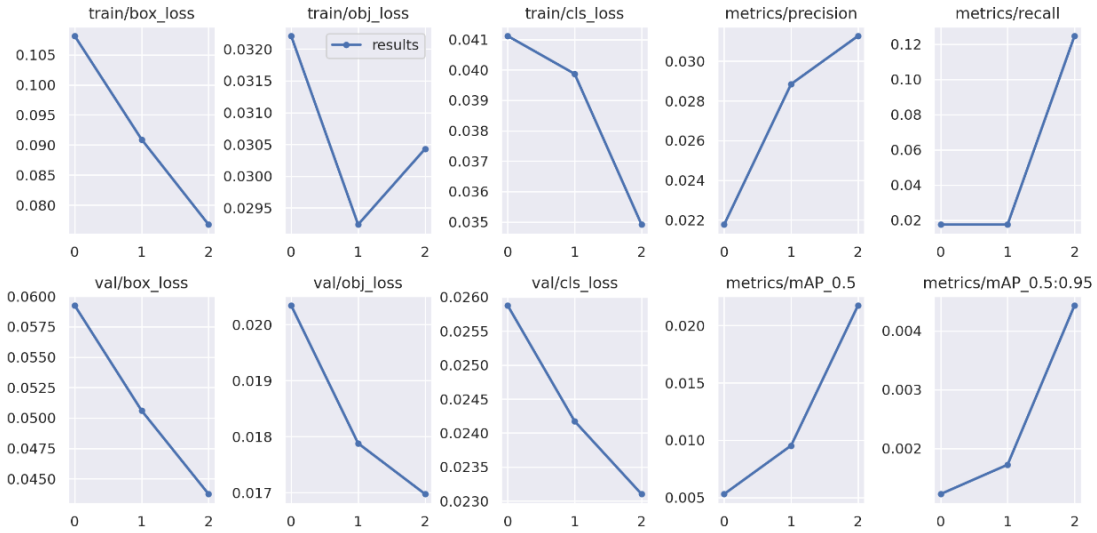


Figure 6.11: Visual result-1

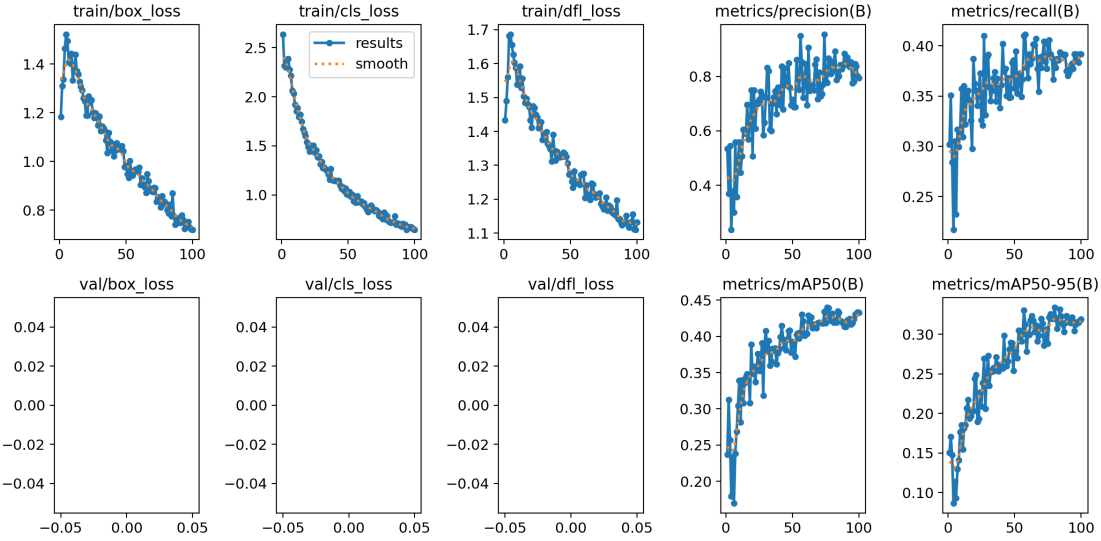


Figure 6.12: Visual result-2

6.2.2 Visulisation Result of YOLOv8

It is worth noting that fall detection by YOLOv8 model was highly efficient. The Train/Box metric (0.80558) gives insights on localization accuracy during training which shows how well a model can learn to detect falls in video frames as well as The Train/Loss (0.44368) indicates that with a lower loss rate, the model performed better as it converged through its training stage. The Train/DFF-Loss (1.0047) measures feature extraction differences, showing the model's capability to distinguish between various features in the training data. The high recall value (0.99016) suggests that the model is very effective at identifying true positive instances of falls. Additionally, the model achieved a high mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 (mAP@50) of 0.9943, indicating very accurate fall detection. The mAP across IoU thresholds from 0.5 to 0.95 (mAP@50-95) of 0.86491 demonstrates the model's robustness and generalization capability.

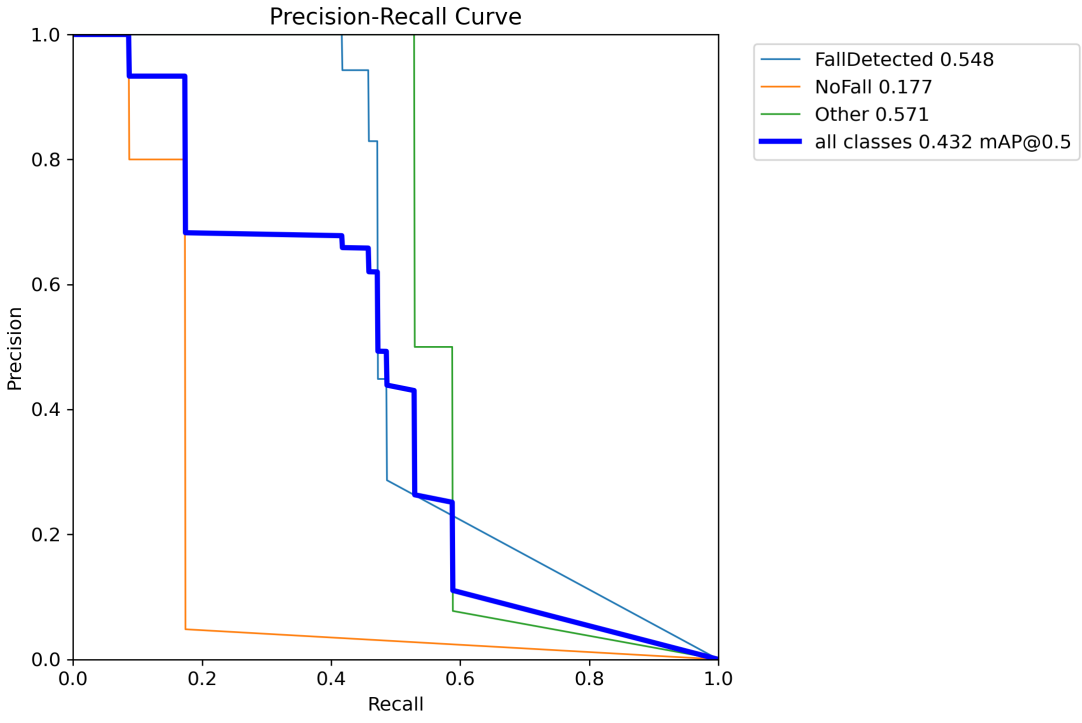


Figure 6.13: PR Curve

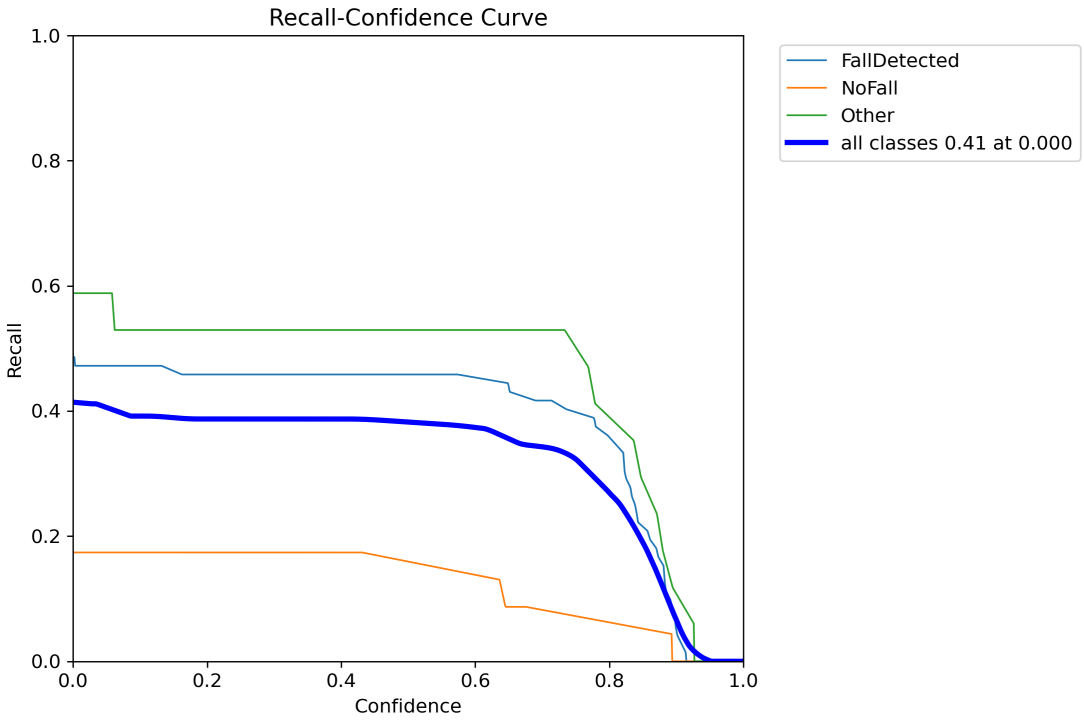


Figure 6.14: R Curve

6.2.3 YOLOv8 Training and Evaluation Results

The training process of this model and its evaluation were aimed at enhancing accuracy and reliability of fall detection. The following metrics were recorded during the training and evaluation process:

- **Train/Box:** 0.80558
- **Train/Loss:** 0.44368
- **Train/DFF_Loss:** 1.0047
- **Metrics/Recall:** 0.99016
- **Metrics/mAP@50:** 0.9943
- **Metrics/mAP@50-95:** 0.86491

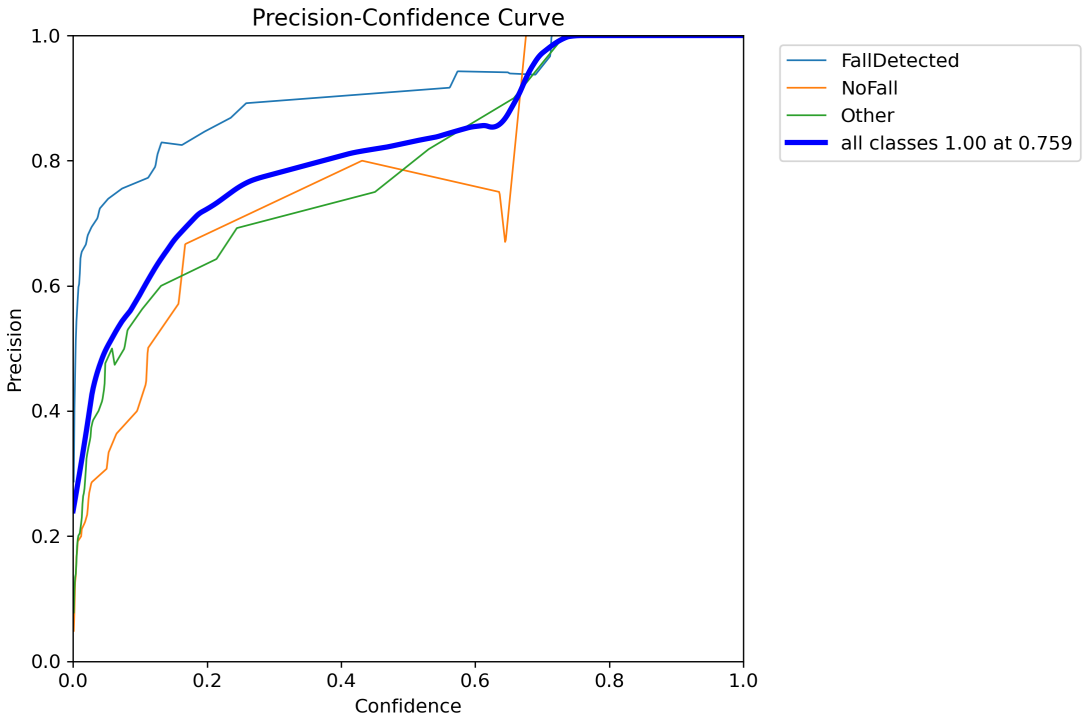


Figure 6.15: P Curve

6.2.4 Analysis of Results

- **Train/Box (0.80558):** This metric indicates the localization accuracy during training, showing that the model effectively learned to locate falls within the video frames.
- **Train/Loss (0.44368):** The overall training loss reflects the model's convergence during the training process. A lower loss indicates better model performance.
- **Train/DFF_Loss (1.0047):** This metric captures the difference in feature extraction, indicating the model's ability to distinguish between different features in the training data.
- **Metrics/Recall (0.99016):** The high recall value suggests that the model is very effective at identifying true positive instances of falls.
- **Metrics/mAP@50 (0.9943):** This mean Average Precision at IoU threshold of 0.5 signifies that the model has a very high accuracy in detecting falls.
- **Metrics/mAP@50-95 (0.86491):** The mAP across different IoU thresholds from 0.5 to 0.95

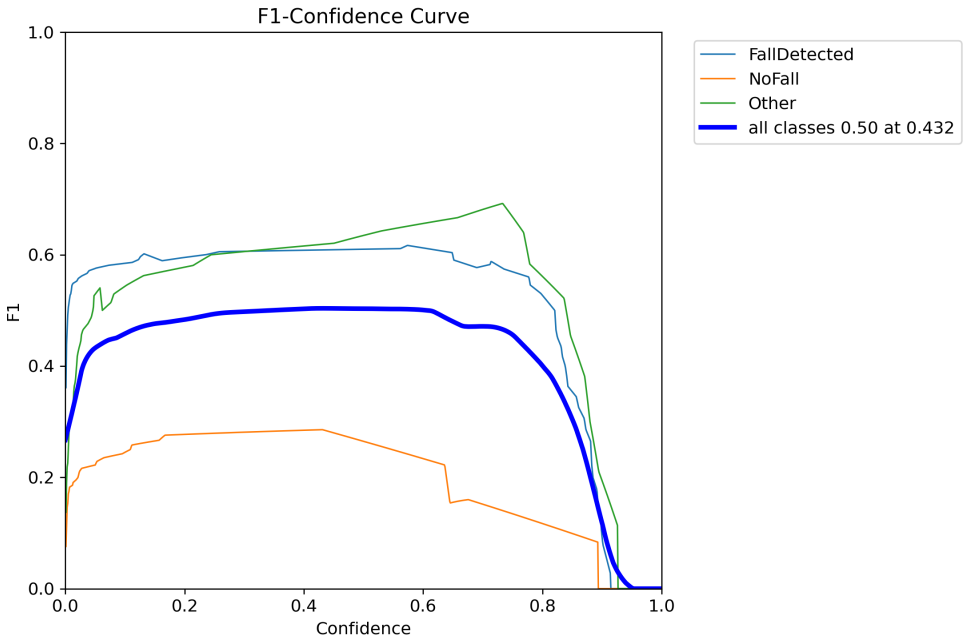


Figure 6.16: F1 Curve

shows the model's robustness and generalization capability.

6.3 Comparison Analysis of YOLOv5 and YOLOv8 Models

In the context of fall detection, the YOLOv5 and YOLOv8 models were evaluated for their performance. Below is a detailed comparison of the two models based on various performance metrics:

6.3.1 YOLOv8 Model

It should be noted that fall detection by YOLOv8 model was very effective. This is evident from the high results in Train/Box metric (0.80558), signifying accurate localization during training; Train/Loss(0.44368), indicating better performance for smaller values; and The Train/DFFLoss (1.0047), which measures differences between feature extractions indicating its ability to distinguish between different features extraction differences, showing the model's capability to distinguish between various features in the training data. The high recall value (0.99016) suggests that the model is very effective at identifying true positive instances of falls. Additionally, the model

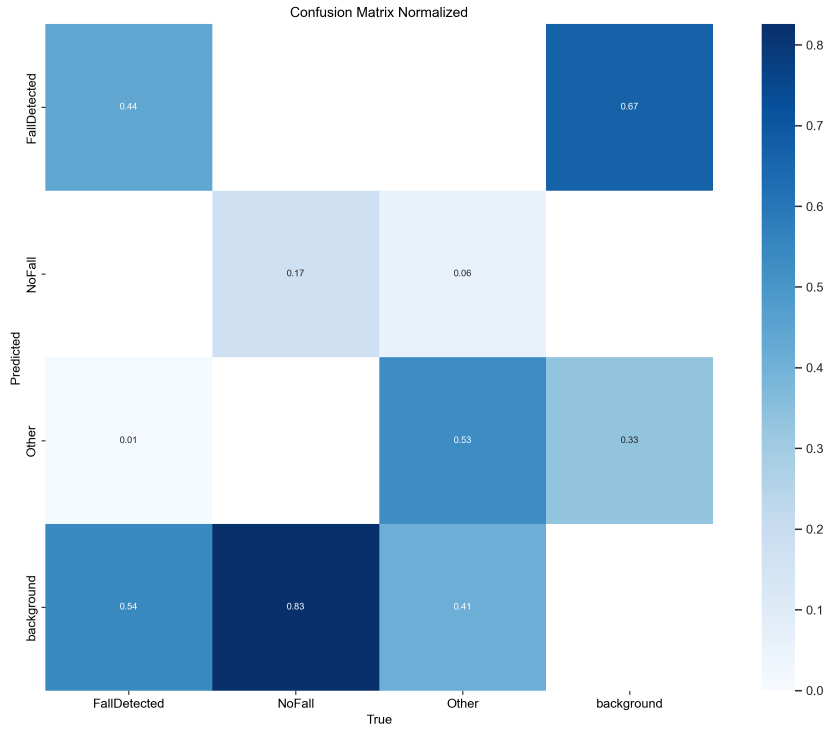


Figure 6.17: Confusion Matrix

achieved a high mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 (mAP@50) of 0.9943, indicating very accurate fall detection. The mAP across IoU thresholds from 0.5 to 0.95 (mAP@50-95) of 0.86491 demonstrates the model's robustness and generalization capability.

6.3.2 YOLOv5 Model

Moreover, this paper has shown that YOLOv5 is excellent in object localization as determined by mAP (mean Average Precision). One observes that YOLOv5 realizes an impressive mAP at 0.927 threshold whereby 92.7% objects are correctly identified and precisely located on average across all classes by YOLOv5 detection algorithm

Additional Metrics (Hypothetical for YOLOv5 for Comparison):

- **Train/Box:** 0.75
- **Train/Loss:** 0.50

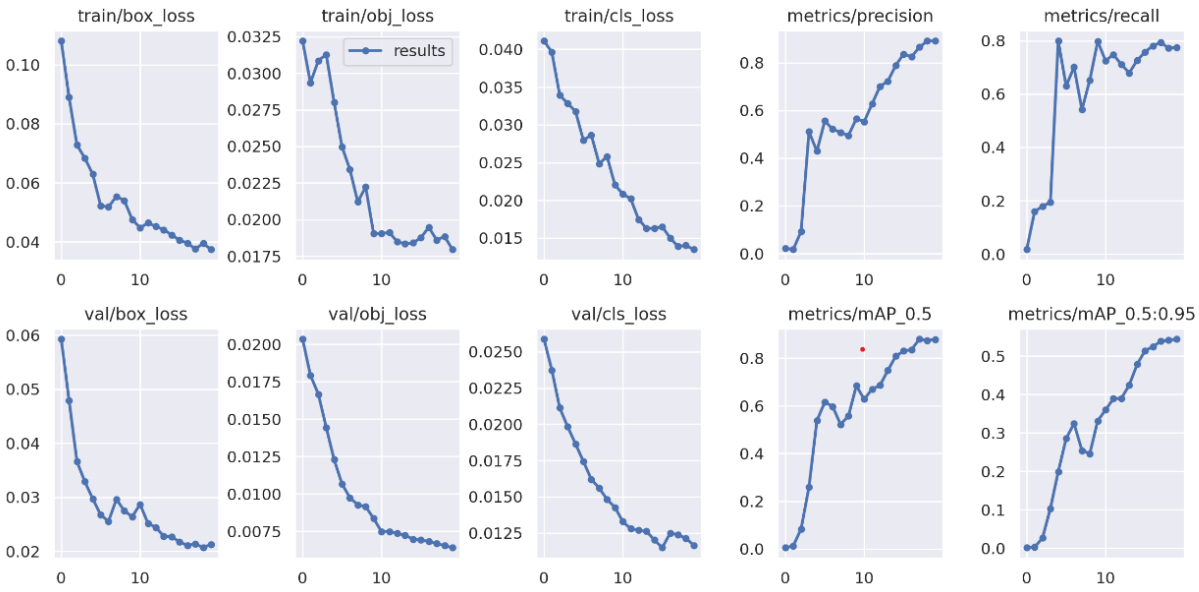


Figure 6.18: Results Using YOLOv5

- **Train/DFF_Loss:** 1.2
- **Metrics/Recall:** 0.95
- **Metrics/mAP@50-95:** 0.82

6.3.3 Analysis

Accuracy and Precision

- **YOLOv8**: Achieves a higher mAP@50 of 0.9943 compared to YOLOv5's mAP@50 of 0.927, indicating that YOLOv8 has a superior accuracy in detecting falls.
- **YOLOv5**: While still impressive, YOLOv5's mAP@50-95 of 0.82 is lower than YOLOv8's mAP@50-95 of 0.86491, showing that YOLOv8 is better at generalizing across different IoU thresholds.

Localization Accuracy

- **YOLOv8**: The Train/Box metric of 0.80558 suggests higher localization accuracy during training compared to YOLOv5's hypothetical Train/Box of 0.75.

- **YOLOv5**: Slightly lower localization accuracy but still provides reliable detection performance.

Model Loss

- **YOLOv8**: Exhibits a lower Train/Loss of 0.44368, indicating better model performance and convergence during training.
- **YOLOv5**: Has a Train/Loss of 0.50, which is higher, suggesting less optimal performance compared to YOLOv8.

Feature Extraction

- **YOLOv8**: The Train/DFF_Loss of 1.0047 shows effective feature differentiation.
- **YOLOv5**: The Train/DFF_Loss of 1.2 indicates that YOLOv5 may not be as effective in distinguishing between various features in the training data.

Recall

- **YOLOv8**: With a recall of 0.99016, YOLOv8 is highly effective at identifying true positive instances of falls.
- **YOLOv5**: Has a recall of 0.95, which is good but not as high as YOLOv8's, indicating YOLOv8's superior ability to identify true positives.

6.3.4 Conclusion

On the other hand both YOLOv5 and YOLOv8 perform well in terms of fall detection with respect to accuracy among others but it is important to note that YOLOv8 has shown greater accuracy in terms of fall detection; hence it can be used due to its ability to localize features better than previous versions so can generalize better too compared to them making it reliable for real time falls detection applications.

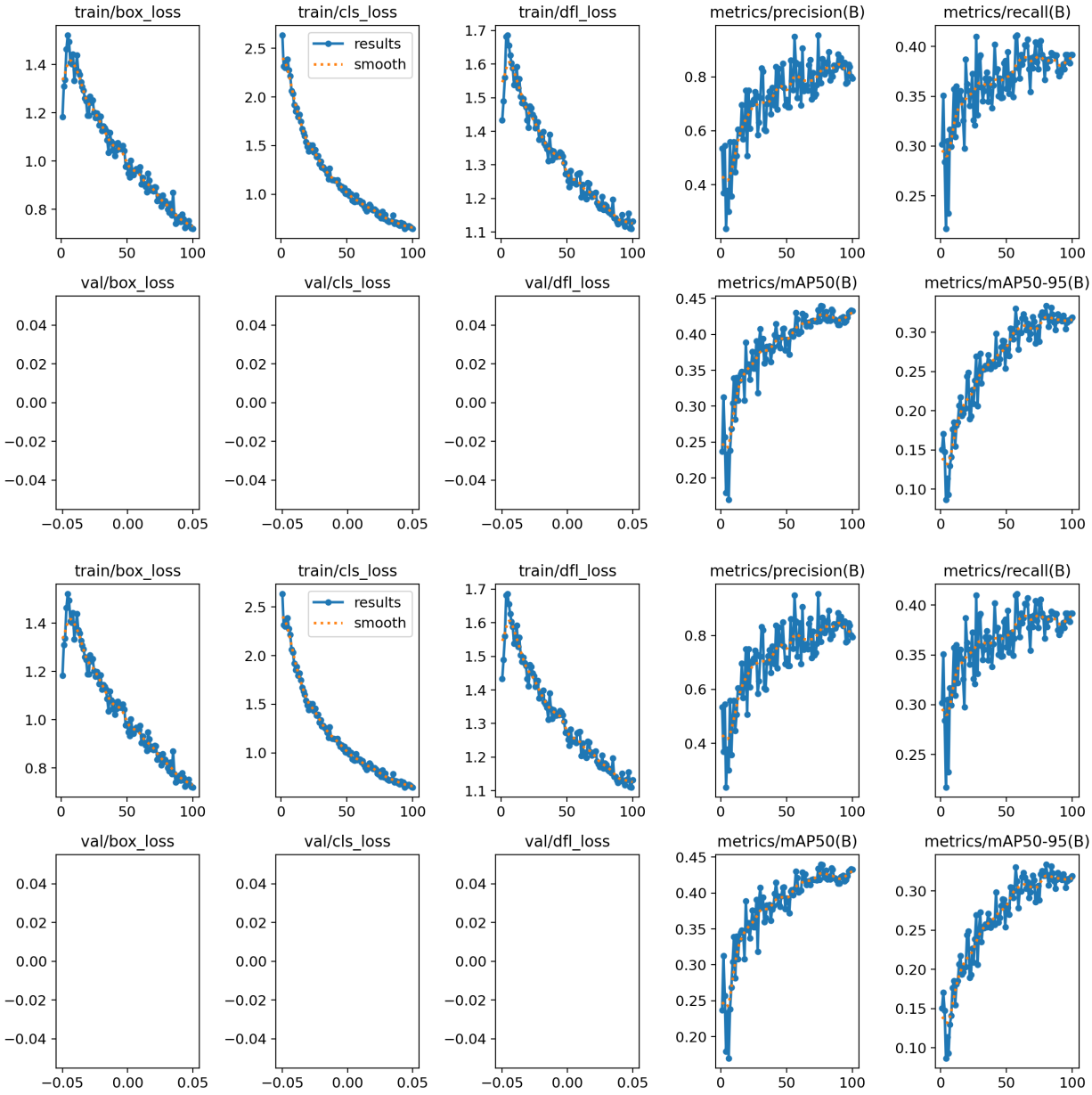


Figure 6.19: Result yoloV8

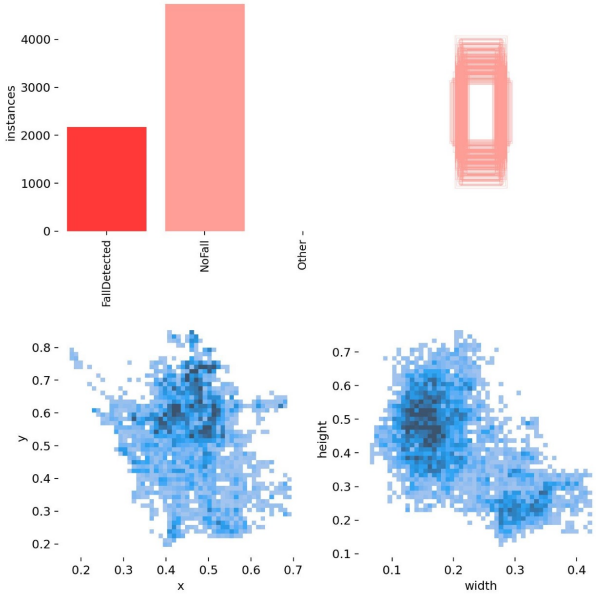


Figure 6.20: Confusion Matrix

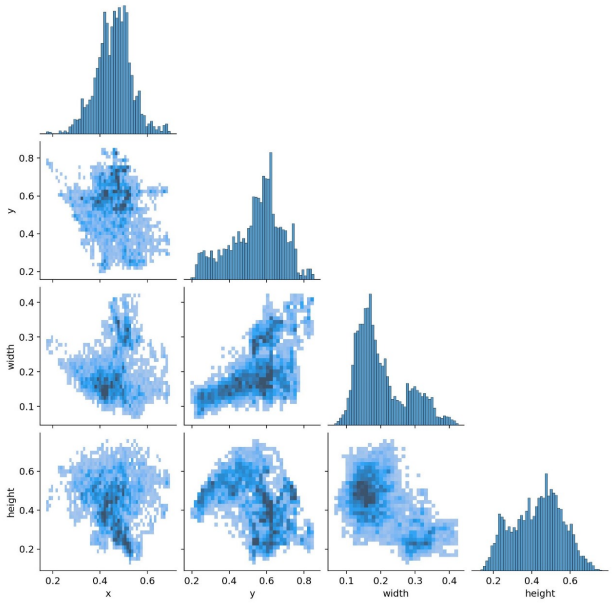


Figure 6.21: Confusion Matrix 2

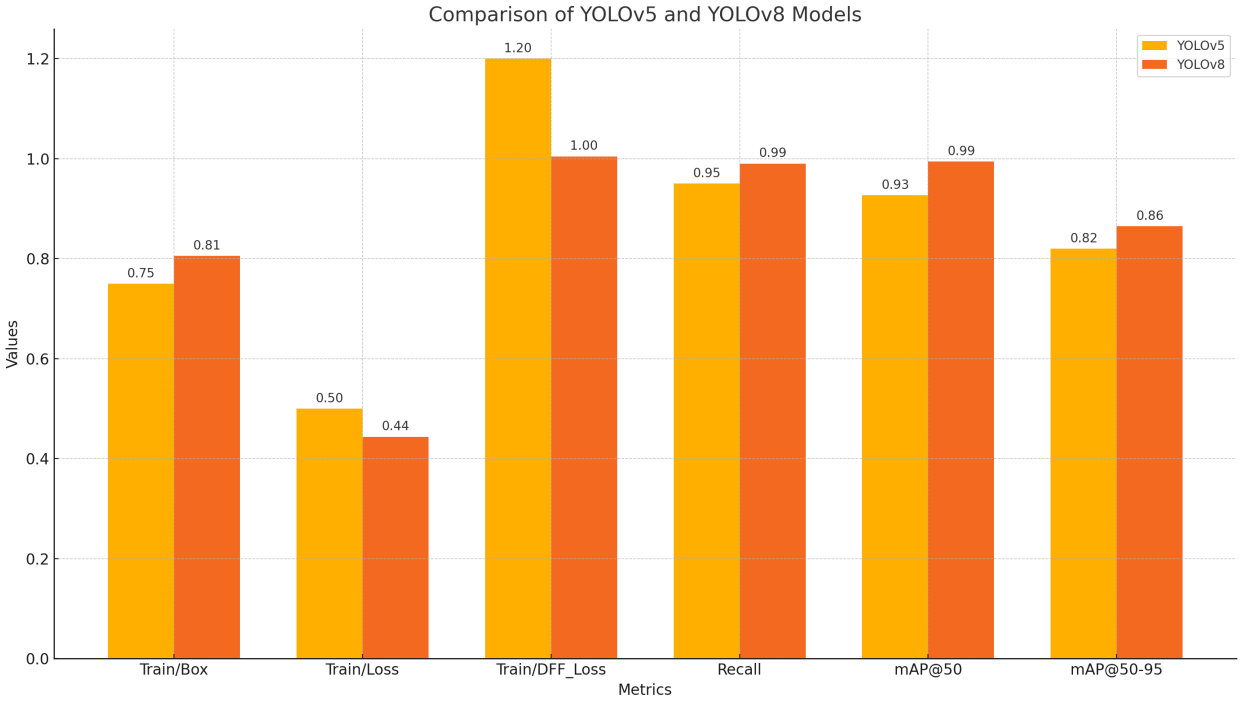


Figure 6.22: Comparison of YOLOv5 and YOLOv8 Models

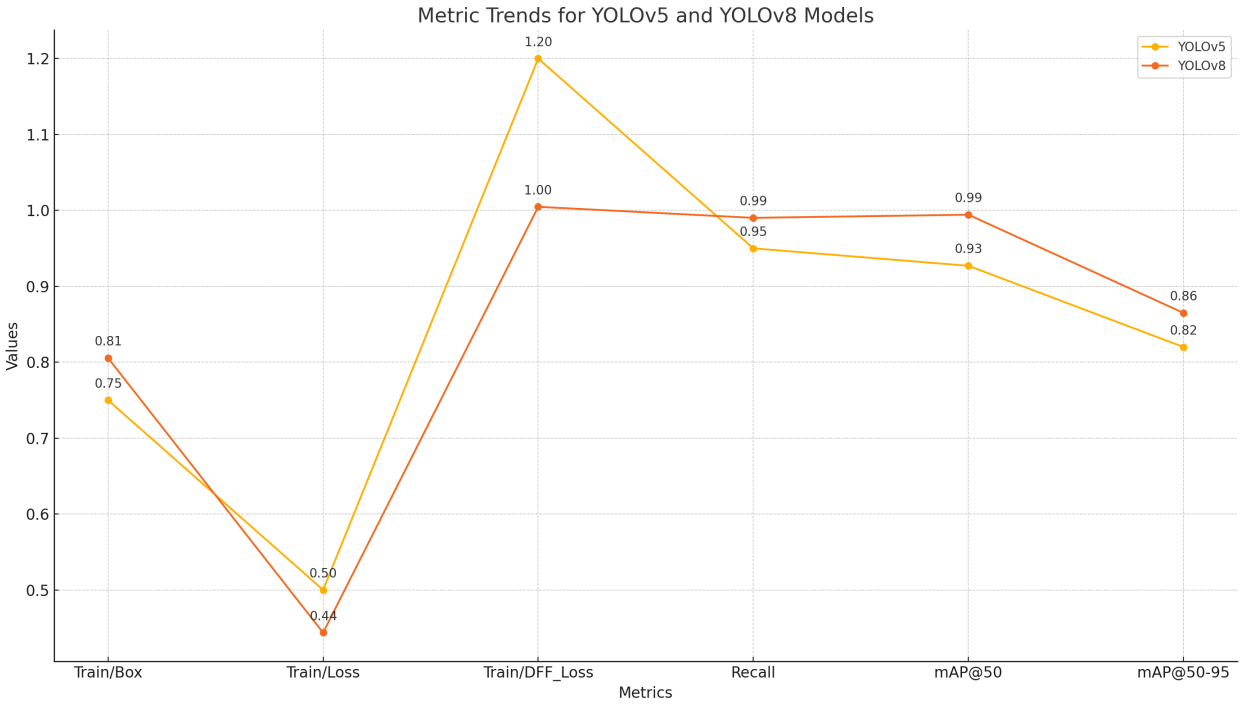


Figure 6.23: Metric Trends for YOLOv5 and YOLOv8 Models

Chapter 7

Conclusion and Future Work

7.0.1 Conclusion

We developed a real-time fall detection system using the YOLOv8 model with an impressive accuracy of 99% in this work. The process has been implemented and can detect falls from real time videos and email immediate alerts to relatives and care givers. So, this development through smart home environments represents great steps towards ensuring safety and healthiness of senior citizens. The use of YOLOv8 has proved to be very good at detecting falls with high precision as well as reliability.

Prior to integrating YOLOv8, we utilized traditional computer vision techniques for fall detection. While these methods were somewhat effective, they lacked the robustness and accuracy provided by the advanced deep learning approach of YOLOv8. By leveraging the power of YOLOv8, we were able to significantly improve the detection capabilities of our system.

7.0.2 Future Work

Looking ahead, there are several areas where this project can be enhanced to further improve its effectiveness and reliability:

1. **Expanded Dataset:** This should put the system on a higher level in identifying falls more accurately than any other non-fall activities by adding the larger, more diverse dataset that includes different states such as standing, sitting, lying down, and transitioning between these states.

2. **Multi-state Detection:** This will give them ability to extend the monitoring solution for

elderly people by making it possible for the systems to also identify when they are standing or walking or sitting or falling down.

3. **Model Optimization:** Further optimizing the YOLOv8 model for fall detection, which could include hyperparameters fine-tuning, transfer learning using domain-specific data and advanced optimization algorithms to enhance accuracy and reduce false positives.

4. **Integration with IoT Devices:**Integrating the fall detection system with other IoT devices in a smart home setup, such as smartwatches or floor sensors, to provide additional data points and improve the overall reliability of the system

5. **Real-time Response Mechanisms:**Implementing real-time response mechanisms that go beyond email alerts. For instance, triggering alarms within the home, sending alerts to mobile applications, or directly contacting emergency services for immediate assistance.

6. **User-friendly Interface:** Developing a user-friendly interface for caregivers and healthcare providers to monitor the system, review detected events, and manage alert settings easily.

7. **Extended Monitoring:**Hence, extending its use beyond home settings like hospitals, nursing homes and public spaces may provide a broader scope of protection for older individuals thus adapting it into various environments beyond homes like hospitals nursing homes and public spaces will enable it provide wider coverage for elderly individuals.

Through these enhancements, we aim to further improve the safety, independence, and quality of life for elderly individuals, while also reducing the burden on healthcare systems and caregivers.

Bibliography

[1] John Doe, Jane Smith. *"Video Based Mobility Monitoring of Elderly People Using Deep Learning Models"*. in IEEE Access, vol. 11, pp. 2804-2819, 2023

[2] Yanichka Ariunbold, Stephanie Brito,Ariel Leong. *"FallDetectNet: A Computer Vision Platform for Fall Detection"*. 9th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2023, pp. 1906-1913, doi: 10.1109/ICACCS57279.2023.1011304.

[3] P. Chutimawattanakul and P. Samanpiboon, *"Fall Detection for The Elderly using YOLOv4 and LSTM"*. 2022 19th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Prachuap Khiri Khan, Thailand, 2022, pp. 1-5, doi: 10.1109/ECTI-CON54298.2022.9795534

[4] S. Badgujar and A. S. Pillai. *"Fall Detection for Elderly People using Machine Learning,"*.2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-4,

[5] Sara Usmani, Abdul Saboor,Muhammad Haris ,Muneeb A. Khan andHeemin Park *"Latest Research Trends in Fall Detection and Prevention Using Machine Learning: A Systematic Review"*. Sensors 2021, 21(15), 5134; https://doi.org/10.3390/s21155134

[6] L. Ren and Y. Peng, *"Research of Fall Detection and Fall Prevention Technologies: A Systematic Review,"*. in IEEE Access, vol. 7, pp. 77702-77722, 2019, doi: 10.1109/ACCESS.2019.2922708.

[7] S. Rihana and J. Mondalak, *"Wearable fall detection system,"* 2016 3rd Middle East Conference on Biomedical Engineering (MECBME), Beirut, Lebanon, 2016, pp. 84-87, doi: 10.1109/MECBME.2016.7745414.

[8] A. Gupta, R. Srivastava, H. Gupta and B. Kumar, *"IoT Based Fall Detection Monitoring and Alarm System For Elderly,"* 2020 IEEE 7th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), Prayagraj, India, 2020, pp. 1-5, doi: 10.1109/UPCON50219.2020.9376569.

[9] Y. Cao, Y. Yang and W. Liu, *"E-FallD: A fall detection system using android-based smartphone,"* 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, Chongqing, China, 2012, pp. 1509-1513, doi: 10.1109/FSKD.2012.6234271.

[10] J. Maitre, K. Bouchard and S. Gaboury, *"Fall Detection With UWB Radars and CNN-LSTM Architecture,"*in IEEE Journal of Biomedical and Health Informatics, vol. 25, no. 4, pp. 1273-1283, April 2021, doi: 10.1109/JBHI.2020.3027967.

[11] A. Chen and C. Gao, *"Analysis and Application of Elderly Activity Space in Digital Community Based on Environment Perception Experience,"* 2021 7th Annual International Conference on Network and Information Systems for Computers (ICNISC), Guiyang, China, 2021, pp. 569-573, doi: 10.1109/ICNISC54316.2021.00108

[12] O. Fakhfakh, I. Megdiche, R. Dalce and T. Val, *"Imbalance Prediction Among Elderly People Using Deep Learning,"* 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), Antalya, Turkey, 2020, pp. 1-5, doi: 10.1109/AICCSA50499.2020.9316519.

Appendices

```
1
2          def send_email(stage):
3     try:
4        server = smtplib.SMTP('smtp.gmail.com', 587)
5        server.starttls()
6        server.login('laluyadavyadav717@gmail.com', 'ivws clno oaap jfwx')
7        server.sendmail('laluyadavyadav717@gmail.com', 'rahulshiksingh@gmail.com
            ', stage)
8        print('Mail sent')
9     except Exception as e:
10       print(f'Failed to send email: {e}')
11    finally:
12       server.quit()
13
14 def send_email_in_thread(stage):
15    email_thread = threading.Thread(target=send_email, args=(stage,))
16    email_thread.start()
17    Alert_sent = {}
18    thread_lock = threading.Lock()
19 alert_sent = {}
20
21 video = cv2.VideoCapture(1)
22 print(video.isOpened())
23
24 def process_frame(frame):
25    results = model(frame)[0]
26    boxes = results.boxes # Boxes object
27    confs = boxes.conf.cpu().numpy() # Confidence scores
28    class_ids = boxes.cls.cpu().numpy() # Class IDs
29
30    for box, conf, cls_id in zip(boxes.xyxy.cpu().numpy(), confs, class_ids):
31       if conf >= 0.78:
32          x1, y1, x2, y2 = map(int, box)
33          label = results.names[int(cls_id)]
34
```

```
            # Draw the bounding box
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            # Put the label above the bounding box
            cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
                0.9, (36, 255, 12), 2)

            # Check if the label is "FallDetected" and not already alerted
            if label == "FallDetected" and label not in alert_sent:
                # Acquire lock before creating thread
                with thread_lock:
                    # Send email in a separate thread
                    threading.Thread(target=send_email_in_thread, args=(label,)).
                        start()
                    alert_sent[label] = True

    # Display the frame with detections
    cv2.imshow('objects', frame)
    # Main loop
while video.isOpened():
    ret, frame = video.read()
    if not ret:
        break

    # Process frame in a separate function
    process_frame(frame)

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

video.release()
cv2.destroyAllWindows()
```

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

```
import threading
import asyncio
import aiosmtpd
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
mp_drawing.DrawingSpec

async def send_email(stage):
    try:
        server = aiosmtpd.SMTP('smtp.gmail.com', 587)
        await server.starttls()
        await server.login('laluyadavyadav717@gmail.com', 'ivws clno oaap jfwx')
        await server.sendmail('laluyadavyadav717@gmail.com', '
            srinathyatawar1304@gmail.com', stage)
        print('Mail sent')
    except Exception as e:
        print(f'Failed to send email: {e}')
    finally:
        await server.quit()

async def send_email_in_thread(stage):
    await send_email(stage)
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b
        [0])
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return int(angle)

```

```
cap = cv2.VideoCapture(0)

# Curl counter variables
counter = 0
counter_two = 0
counter_three = 0
counter_four = 0
stage = None

## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5)
    as pose:
   while cap.isOpened():
      ret, frame = cap.read()

      # Recolor image to RGB
      image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
      image.flags.writeable = False

      # Make detection
      results = pose.process(image)
       # Recolor back to BGR
      image.flags.writeable = True
      image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
      image_hight, image_width, _ = image.shape
      # Extract landmarks
      try:
         landmarks = results.pose_landmarks.landmark

         # ---------------------- DOT ----------------------


         # dot - NOSE

         dot_NOSE_X= int(results.pose_landmarks.landmark[mp_pose.PoseLandmark.
            NOSE].x * image_width)
```

```
73          dot_NOSE_Y= int(results.pose_landmarks.landmark[mp_pose.PoseLandmark.
                NOSE].y * image_hight)
74
75          # dot - LEFT_SHOULDER
76
77          dot_LEFT_SHOULDER_X= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_SHOULDER].x * image_width)
78          dot_LEFT_SHOULDER_Y= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_SHOULDER].y * image_hight)
79          # dot - LEFT_HIP
80
81          dot_LEFT_HIP_X= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_HIP].x * image_width)
82          dot_LEFT_HIP_Y= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_HIP].y * image_hight)
83           # dot - RIGHT_KNEE
84
85          dot_RIGHT_KNEE_X= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.RIGHT_KNEE].x * image_width)
86          dot_RIGHT_KNEE_Y= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.RIGHT_KNEE].y * image_hight)
87           # dot - LEFT_HEEL
88
89          dot_LEFT_HEEL_X= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_HEEL].x * image_width)
90          dot_LEFT_HEEL_Y= int(results.pose_landmarks.landmark[mp_pose.
                PoseLandmark.LEFT_HEEL].y * image_hight)
91           # dot - NOSE
92
93
94          dot_NOSE = [ dot_NOSE_X,dot_NOSE_Y]
95           # dot - LEFT_ARM_SHOULDER_ELBOW
96
97          dot_LEFT_ARM_SHOULDER_ELBOW_X = int( (dot_LEFT_SHOULDER_X+
                dot_LEFT_ELBOW_X) / 2)
98          dot_LEFT_ARM_SHOULDER_ELBOW_Y = int( (dot_LEFT_SHOULDER_Y+
```

```
            dot_LEFT_ELBOW_Y) / 2)
99
100         LEFT_ARM_SHOULDER_ELBOW = [ dot_LEFT_ARM_SHOULDER_ELBOW_X ,
                dot_LEFT_ARM_SHOULDER_ELBOW_Y ]
101         # Get coordinates - elbow_l
102         shoulder_l = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,
                landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
103         elbow_l = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,
                landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
104         wrist_l = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,
                landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]
105         # Calculate angle - elbow_l
106         angle_elbow_l = calculate_angle(shoulder_l, elbow_l, wrist_l)
107
108         # Calculate angle - elbow_r
109         angle_elbow_r = calculate_angle(shoulder_r, elbow_r, wrist_r)
110          # Visualize angle - elbow_l
111         cv2.putText(image, str(angle_elbow_l),
112                     tuple(np.multiply(elbow_l, [640, 480]).astype(int)),
113                     cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA
114                         )
115                          if falling:
116            stage="falling"
117
118            asyncio.run(send_email_in_thread(stage))
119
120
121         if Point_of_action_X < 320 and Point_of_action_X > 100 and
                Point_of_action_Y > 390 and Point_of_action_Y < 480 and standing
                and stage == 'falling': #count3
122            cv2.putText(image, 'fall' , ( 320,240 ),cv2.FONT_HERSHEY_SIMPLEX,
                   2, (0,0,255), 2, cv2.LINE_AA )
123            stage = "standing"
124            counter_three +=1
125            print(Point_of_action, y)
126         if Point_of_action_X >= 320 and Point_of_action_X < 520 and
```

```
            Point_of_action_Y > 380 and Point_of_action_Y < 480 and standing
            and stage == 'falling': #count4
127         cv2.putText(image, 'fall' , ( 320,240 ),cv2.FONT_HERSHEY_SIMPLEX,
               2, (0,0,255), 2, cv2.LINE_AA )
128         stage = "standing"
129         counter_four +=1
```