Real-Time Human Motion Analysis for Public Safety Using Machine Learning

Kayes Rasha Md Samaun
ID: 2019–1–60–026

Ripon Chandra Saha
ID: 2020-1-60-233

Sumaya Akter
ID: 2019-1-67-056

Jumaina Hossain Esha
ID: 2020-3-60-061

Supervisor:
Dr. Anisur Rahman
Associate Professor
Dept. of Computer Science & Engineering
East West University

A Capstone project report submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh

February, 2025

Declaration

We, **Kayes Rasha Md Samaun, Ripon Chandra Saha, Sumaya Akter and Jumaina Hossain Esha** hereby, declare that the work presented in this capstone project report is the outcome of the investigation performed by us under the supervision of **Dr. Anisur Rahman**, Associate Professor, Department of Computer Science and engineering, East West University. We also declare that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma, except for publication.

Countersigned	Signature
. . . .	. . . .
Dr. Anisur Rahman	**Kayes Rasha Md Samaun**
Supervisor	**2019-1-60-026**

Signature

. . . .

Ripon Chandra Saha

2020-1-60-233

Signature

. . . .

Sumaya Akter
2019-1-67-056

Signature

. . . .

Jumaina Hossain Esha
2020-3-60-061

Letter of Acceptance

The capstone project report entitled "Real-Time Human Motion Analysis for Public Safety Using Machine Learning" is submitted by Kayes Rasha Md Samaun, Ripon Chandra Saha, Sumaya Akter and Jumaina Hossain Esha to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering on (27/02/2025).

Board of Examiners

1. ____

Dr. Anisur Rahman (Capstone Project Supervisor)

Associate Professor
Department of Computer Science and Engineering
East West University

2. ____

Dr. Maheen Islam (Chairperson)

Associate Professor
Department of Computer Science and Engineering
East West University

Abstract

Ensuring public safety through real-time surveillance and rapid emergency response has become increasingly vital in modern urban environments. This study presents a real-time human motion analysis system using machine learning, specifically the YOLOv8 framework, to detect and classify human activities for public safety applications. The system is designed to process live video feeds and identify behaviors.

The research explores advancements in Human Action Recognition (HAR), analyzing traditional machine learning models and their evolution into sophisticated deep learning architectures. A primary challenge addressed is the variation in camera angles, which can impact detection accuracy. To enhance robustness, the system incorporates data augmentation techniques and a refined training dataset, improving generalization across diverse surveillance scenarios. Another critical issue tackled is the detection of occluded actions, for which the study explores multi-sensor data fusion techniques.

The YOLOv8 model was optimized by reducing the number of bottleneck layers in the C2f block, leading to around 10-second reduction in training time per epoch while maintaining high accuracy. The final system achieved 86% mAP@0.5, 82% precision, and 78% recall, demonstrating strong performance across multiple action classes, including walking, standing, sitting, and falling.

The project contributes to the field of real-time motion analysis by improving computational efficiency without compromising accuracy. It provides an intuitive, scalable, and deployable solution for smart surveillance and emergency response systems. Future enhancements include integrating attention mechanisms, model quantization, and deployment on edge devices to further enhance real-time performance and adaptability in real-world applications.

Acknowledgments

As it is true for everyone, we have also arrived at this point of achieving a goal in our life through various interactions with and help from other people. However, written words are often elusive and harbor diverse interpretations even in one’s mother language. Therefore, we would not like to make efforts to find best words to express my thankfulness other than simply listing those people who have contributed to this thesis itself in an essential way. This work was carried out in the Department of Computer Science and Engineering at East West University, Bangladesh.

First of all, we would like to express my deepest gratitude to the almighty for His blessings on us. Next, our special thanks go to our supervisor, “Dr Anisur Rahman”, who gave us this opportunity, initiated us into the field of “Data Science”, and without whom this work would not have been possible. His encouragements, visionaries and thoughtful comments and suggestions, unforgettable support at every stage of our B.Sc. study were simply appreciating and essential. His ability to muddle us enough to finally answer our own question correctly is something valuable what we have learned and we would try to emulate, if ever we get the opportunity.

We would like to thank H M Ahsan Uddin Abir for his overall support and her helpful suggestions in solving tricky technical problems.

Last but not the least, we would like to thank our parents for their unending support, encouragement and prayers.

There are numerous other people too who have shown me their constant support and friendship in various ways, directly or indirectly related to our academic life. We will remember them in our heart and hope to find a more appropriate place to acknowledge them in the future.

Kayes Rasha Md Samaun
February 2025

Ripon Chandra Saha
February 2025

Sumaya Akter
February 2025

Jumaina Hossain Esha
February 2025

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Background

The integration of real-time human motion analysis into public safety frameworks has become increasingly vital in enhancing surveillance and emergency response capabilities. By leveraging machine learning algorithms, these systems can process live video feeds to detect and interpret human activities, thereby enabling prompt identification of suspicious behaviors and facilitating immediate intervention.

Recent advancements in human action recognition (HAR) have significantly improved the accuracy and efficiency of these systems. A comprehensive analysis of current HAR systems highlights the evolution from traditional machine learning techniques to sophisticated deep learning models, which offer superior performance in activity classification tasks[1]. Moreover, the development of multimodal HAR frameworks that incorporate data from various sensors, such as RGB cameras, depth sensors, and inertial measurement units, has further enhanced the robustness and reliability of activity recognition in diverse environments[2].

Despite these technological strides, several challenges persist in the deployment of real-time human motion analysis systems. Variations in camera angles can significantly impact the performance of action detection models, necessitating the design of algorithms capable of generalizing across different viewpoints[3]

Additionally, the scarcity of labeled data for training purposes poses a substantial hurdle, prompting the exploration of few-shot learning approaches to enable accurate action recognition with minimal annotated examples[4]. Furthermore, effectively detecting and tracking human actions in scenarios where body parts are occluded by objects or other individuals remains a complex problem, requiring innovative solutions to ensure reliable performance in real-world applications.

Research Question and Analysis

Public safety concerns have escalated with the increasing number of criminal activities and emergencies in crowded spaces. Traditional surveillance systems rely heavily on human operators, making them prone to fatigue and oversight.
This rise questions like:

Q1: How can human action detection contribute in enhancing public safety and emergency response?
-By deploying real-time action detection systems, authorities can monitor public spaces more effectively, enabling the swift identification of potentially hazardous situations and facilitating prompt interventions.

Q2: What challenges can arise in the deployment of action detection systems in public spaces and how can they be addressed?

- Challenges such as privacy concerns, environmental variability, and technical limitations must be addressed. Solutions may include implementing robust data anonymization techniques, designing adaptable algorithms, and ensuring compliance with legal and ethical standards.

Q3: How can we design action detection models that generalize well across different camera angles?
- Incorporating data augmentation techniques and training on diverse datasets can enhance the model's ability to generalize across different perspectives, thereby improving accuracy in varied surveillance scenarios.

Q4: How can we effectively detect and track human actions when parts of the body are hidden by objects or other people?
- Utilizing information gathered from multiple types of sensors simultaneously can enhance the system's capability to interpret incomplete visual information, ensuring reliable action detection even in occluded scenarios.

Project Objectives

The primary objective of this project is to design a real-time human motion analysis system. The system aims to enable real-time detection by processing video streams instantaneously to identify human activities. Additionally, it aims to focus on activity recognition by accurately classifying various human actions such as walking, standing, sitting and falling. An essential component of the project is anomaly detection, ensuring that unusual behaviors are identified while minimizing false positives and false negatives. The system is intended to be scalable, maintaining optimal performance across multiple video feeds and large public spaces. Furthermore, a user-friendly interface is planned to provide an intuitive monitoring and alerting system for public safety personnel. Ensuring seamless integration with existing surveillance and emergency response infrastructure is another crucial aspect of this project. Lastly, the system is expected to incorporate continuous learning mechanisms, allowing it to self-improve and adapt to new scenarios over time, enhancing its effectiveness and accuracy.

Project Contributions

This project aims to contribute to the field of real-time human action detection by developing a real-time detection framework tailored for public safety applications. It enhances model robustness against camera angle variations using data augmentation and transfer learning techniques. The system can be continuously trained and adapt to new scenarios, improving its accuracy and effectiveness over time.

Project Outlines

This research paper is structured into five main chapters. Chapter 1 provides an introduction to the research, covering the background, problem statement, objectives, and contributions of the project. Chapter 2 presents a review of related works, discussing previous studies and existing methodologies related to human action detection. Chapter 3 outlines the materials and methods used in the research, detailing the dataset selection, preprocessing techniques, machine learning models, and evaluation metrics employed. Chapter 4 presents the results obtained from experiments and discusses the implications of the findings, highlighting the system's effectiveness and challenges encountered. Chapter 5 concludes the study by summarizing key insights, contributions, and potential future research directions. The paper also includes a bibliography that lists all references in IEEE format and an appendix that maps the course and program outcomes relevant to this research.

Chapter 2

Related Works

Overview

This section focuses on the existing and related work related to the topic. In the earlier works, the authors talk about the evolution of real-time human motion analysis, transitioning from traditional motion tracking techniques to advanced AI-driven approaches. Over time, methods have improved in accuracy, efficiency, and real-time performance, benefiting fields like fitness, healthcare, surveillance, and object detection. Studies on posture correction in gym exercises and medical rehabilitation show that AI-based motion tracking can provide real-time feedback, enhancing technique and reducing injury risks. Similarly, surveillance and security applications have leveraged machine learning models like YOLO and CNNs to improve human motion recognition, even under challenging conditions such as occlusion and poor lighting. However, these systems still face limitations, including reliance on high-quality video feeds, computational demands, and difficulties in handling complex movements or environmental variations.

Despite these challenges, AI and deep learning advancements have significantly enhanced real-time tracking and detection capabilities. The integration of metadata with AI has further refined object detection, improving accuracy across applications like autonomous vehicles and augmented reality. However, the increasing use of AI in motion analysis raises ethical and privacy concerns, particularly regarding surveillance and data security. Future developments in computer vision and Edge AI are expected to make these systems more robust, adaptable, and efficient, addressing existing limitations while unlocking new possibilities for real-time motion analysis in public safety, healthcare, and other critical domains.

Summary

Table 1: Related Works

No of Ref	**Source Information**	**Research Objective**	**Problem Addressed**	**Results**	**Limitation**	**Finding**
[5]	Real-Time Posture Correction in Gym Exercises: A Computer Vision-Based Approach for Performance Analysis, Error Classification, and Feedback. (2023)	Investigate the use of computer vision for real-time posture correction during exercise in the gym and its efficacy.	The study tackles the issue of poor posture when working out in the gym, which can result in accidents and less productive workouts.	A YOLOv7-pose system tracks gym posture in real time, providing feedback. Testing showed promise, with UI improvements suggested.	Depending on how hard the workouts are and how well the video feed is captured, the system's efficiency may vary.	Computer vision-based real-time feedback can greatly enhance workout technique and lower the chance of injury.

[6]	Body Posture Detection and Motion Tracking Using Al for Medical (2014)	Develop a tool that benefits users and healthcare professionals by improving the efficacy and safety of medical workouts through the application of AI and image processing.	To ensure good exercise form and tracking, which is essential for efficient rehabilitation and injury prevention, the paper discusses the difficulty of doing so without expert supervision.	Average Precision = 98.61%, Inference = 0.6 Exercises (Accuracy): Biceps Curl=92%, Squats=84%, Push-Ups= 82%	The main drawbacks are that posture detection and motion tracking accuracy can be impacted by external factors and camera quality.	The body position during exercises may be effectively tracked and analyzed by AI and image processing, which can improve exercise process efficiency and provide real-time feedback.
[7]	Real-Time Human Motion Detection & Tracking (2008)	Create a system for tracking and detecting human mobility in real-time to improve accuracy	Accurate motion detection presents challenges with different backdrops, occlusions, and	A real-time system for detecting and tracking human motion in varying lighting conditions.	Real-time performance is impacted by challenges with intricate motions, harsh	The accuracy of detection and tracking performance is enhanced when machine learning and motion

		and resilience in changing contexts.	illuminatio n.		illumination and heavy computing loads.	detection algorithms are combined.
[8]	Human Motion Recognition in Real-time Surveillance System (2010)	Provide a system with better accuracy and fewer processing demands for tracking and detecting human motion in real-time.	Convention al techniques are less accurate in a variety of situations, have large computing costs, and cannot operate in real-time.	Methods for real-time human motion recognition in surveillance, focusing on challenges like occlusion and lighting, and the need for robust algorithms.	Limited testing scenarios, hardware requirement s, scaling issues, and a lack of diversity in test subjects or surrounding s are some of the constraints.	Performance , real-time processing, and robust tracking under occlusion and environment al fluctuations are improved via a novel technique.

[9]	Artificial Intelligence for Object Detection and its Metadata (2023)	Improved accuracy and contextual knowledge can be achieved by combining AI with metadata to improve object detection.	Complex situations and variations are difficult for traditional object detection systems to handle. Overcoming these problems is the aim of metadata integration.	AI and metadata improve object detection by providing context and enhancing accuracy, with applications in autonomous vehicles, surveillance, and augmented reality.	Regulating and deploying ethical and privacy concerns responsibly.	AI models, particularly CNNs, increase the precision and effectiveness of object detection. Several applications benefit from the added context that Metadata offers.

Chapter 3

Materials and Method

Materials

Dataset Collection

The dataset used in this study was sourced from the Roboflow Universe, a publicly available repository for machine learning datasets. The original dataset contained five classes: Fall_Down, Sitting, Standing, Walking, and Drinking. The dataset was divided into training (70%), validation (20%), and test (10%) sets.

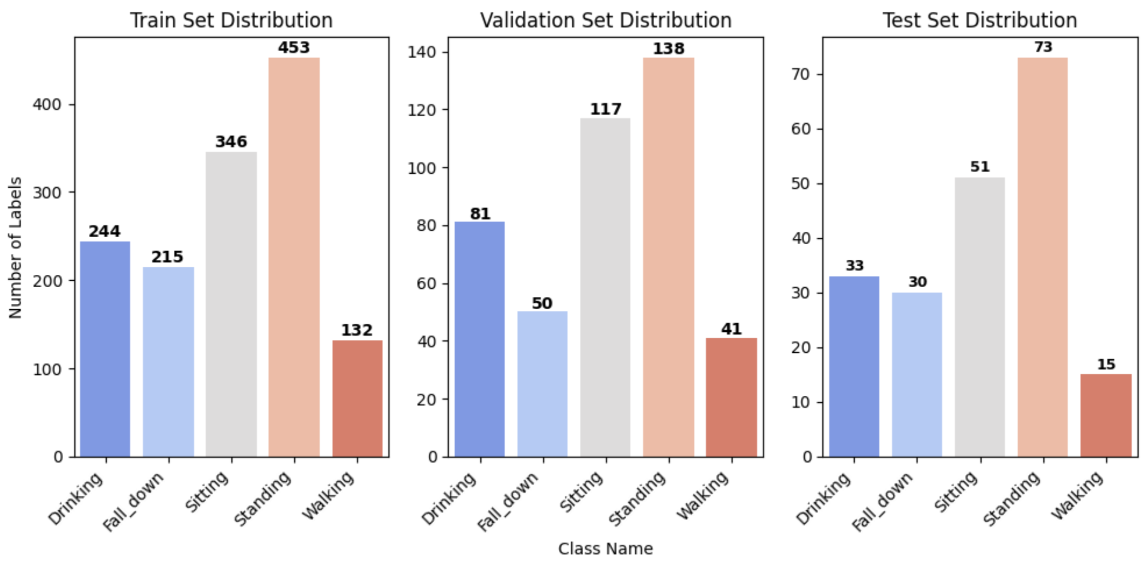


Figure 1: Old Dataset

Total Instances in old Train Set: 1390
Total Instances in old Validation Set: 427
Total Instances in old Test Set: 202

To improve the dataset, Drinking was removed. The refined dataset contained four primary motion categories: Fall_Down, Sitting, Standing, and Walking with updated sample distributions.

Dataset Exploration

The dataset was analyzed for class balance and overall distribution. Initial observations showed a slight class imbalance, with Walking being the most frequent and Standing the least.

Dataset Processing

Roboflow was used for preprocessing and augmentation. Preprocessing steps included Auto-Orient to align images correctly, resize to 640x640 for consistency, Auto-Adjust Contrast using Adaptive Equalization to enhance image quality, and Grayscale conversion to ensure uniformity.

Table 2: Preprocessing

Function	Parameters
Auto-Orient	Applied
Resize	Stretch to 640x640
Auto-Adjust Contrast	Using Adaptive Equalization
Grayscale	Applied

Augmentation techniques were applied to increase dataset diversity and improve generalization. Each training example generated 3 augmented images, with transformations including horizontal flip, rotation between -15° and +15°, shear adjustments of ±10° horizontally and vertically, grayscale applied to 15% of images, saturation variations between -30% and +30%, blur up to 2.9px, and noise affecting up to 1.96% of pixels.

Table 3: Augmentation

Function	Parameters
Flip	Horizontal
Rotation	Between -15° and +15°
Shear	±10° Horizontal, ±10° Vertical
Grayscale	Apply to 15% of images
Saturation	Between -30% and +30%
Blur	Up to 2.9px
Noise	Up to 1.96% of pixels

Dataset Sampling

After preprocessing and augmentation, the dataset was redistributed into training (87%), validation (9%), and test (4%) sets.

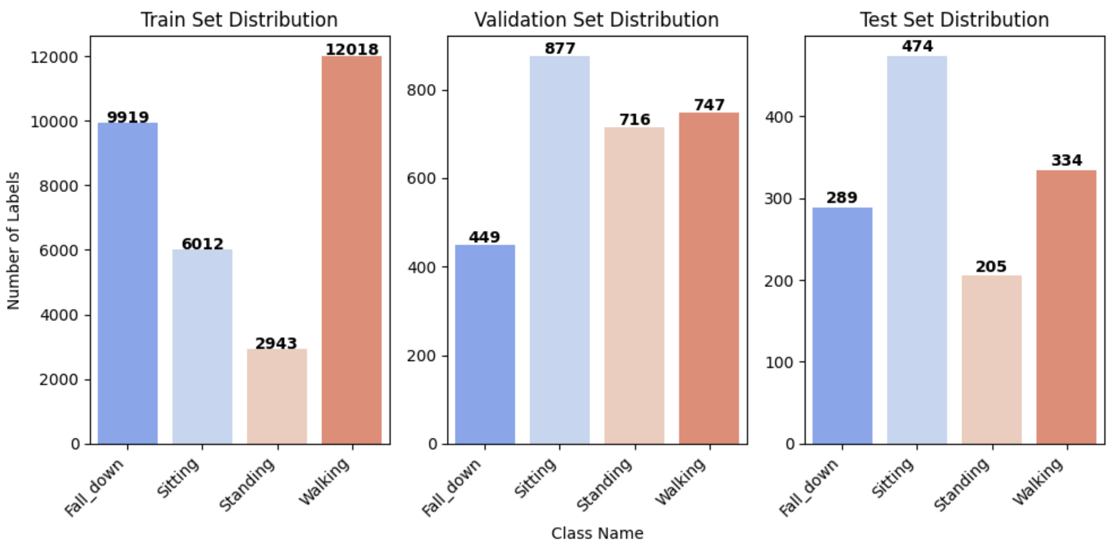


Figure 2: New Dataset

Total Instances in Train Set: 30892
Total Instances in Validation Set: 2789
Total Instances in Test Set: 1302

.

Research Environment and Devices

The model development was conducted using Google Colab, using a T4 GPU for training. Since for this project free version of Google Colab was used, GPU usage was limited. That’s why multiple Google Colab account was created. Following model training, PyCharm was used for further testing and deployment. The trained model was integrated into a Flask-based web application, facilitating real-time human motion analysis.

Method

Proposed Model

In this project, the YOLOv8 architecture was utilized to develop an optimized object detection model. YOLOv8 consisting of three main components: Backbone, Neck, and Head.

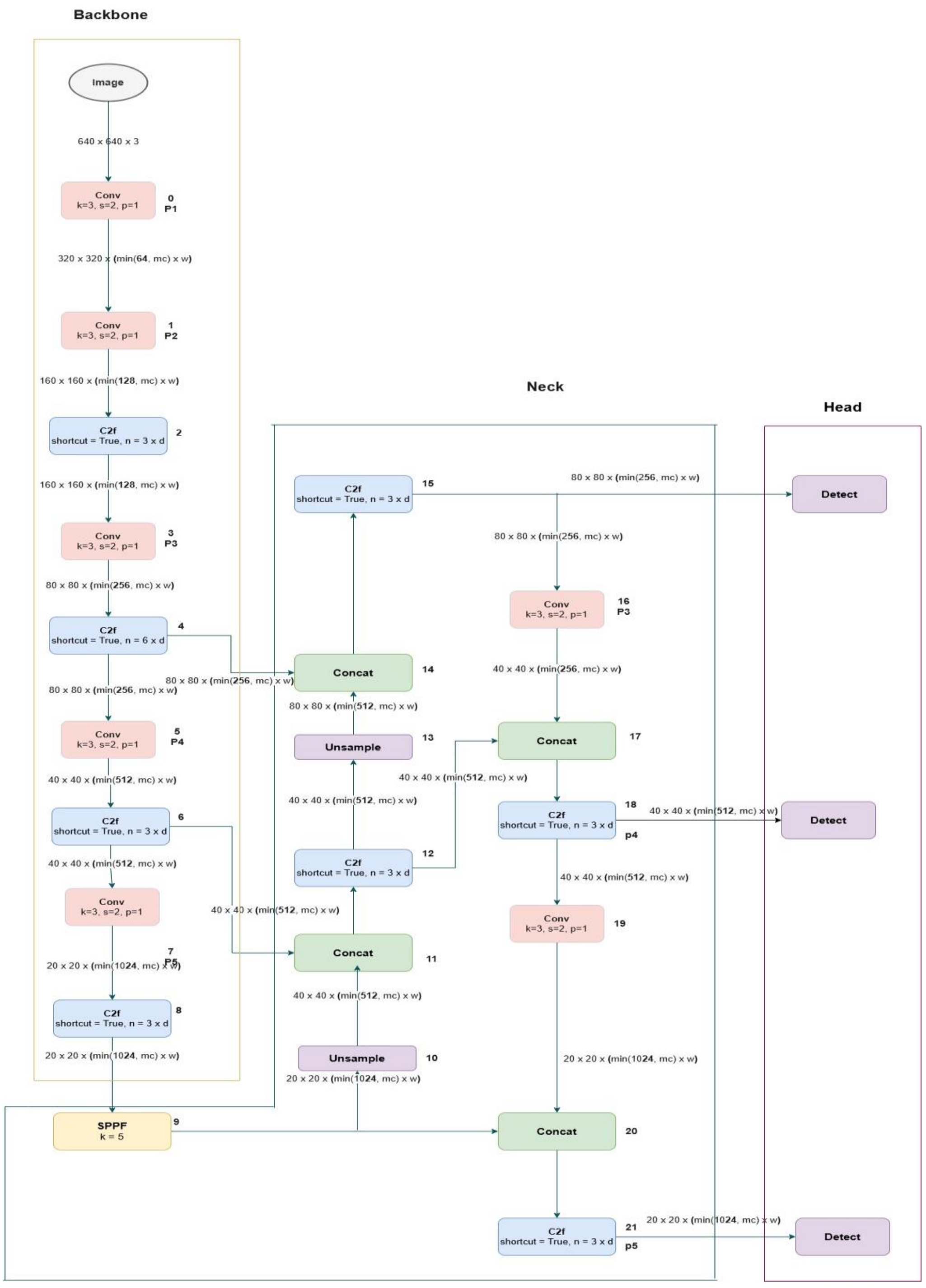

Backbone
Image
640 x 640 x 3
Conv
k=3, s=2, p=1
0
P1
320 x 320 x (min(64, mc) x w)
Conv
k=3, s=2, p=1
1
P2
160 x 160 x (min(128, mc) x w)
C2f
shortcut = True, n = 3 x d
2
160 x 160 x (min(128, mc) x w)
Conv
k=3, s=2, p=1
3
P3
80 x 80 x (min(256, mc) x w)
C2f
shortcut = True, n = 6 x d
4
80 x 80 x (min(256, mc) x w)
Conv
k=3, s=2, p=1
5
P4
40 x 40 x (min(512, mc) x w)
C2f
shortcut = True, n = 3 x d
6
40 x 40 x (min(512, mc) x w)
Conv
k=3, s=2, p=1
7
P5
20 x 20 x (min(1024, mc) x w)
C2f
shortcut = True, n = 3 x d
8
20 x 20 x (min(1024, mc) x w)
SPPF
k = 5
9
Neck
C2f
shortcut = True, n = 3 x d
15
80 x 80 x (min(256, mc) x w)
80 x 80 x (min(256, mc) x w)
Conv
k=3, s=2, p=1
16
P3
40 x 40 x (min(256, mc) x w)
Concat
14
80 x 80 x (min(256, mc) x w)
80 x 80 x (min(512, mc) x w)
Unsample
13
Concat
17
40 x 40 x (min(512, mc) x w)
40 x 40 x (min(512, mc) x w)
C2f
shortcut = True, n = 3 x d
18
p4
40 x 40 x (min(512, mc) x w)
C2f
shortcut = True, n = 3 x d
12
40 x 40 x (min(512, mc) x w)
Conv
k=3, s=2, p=1
19
40 x 40 x (min(512, mc) x w)
Concat
11
40 x 40 x (min(512, mc) x w)
Unsample
10
20 x 20 x (min(1024, mc) x w)
20 x 20 x (min(1024, mc) x w)
Concat
20
C2f
shortcut = True, n = 3 x d
21
p5
20 x 20 x (min(1024, mc) x w)
Head
Detect
Detect
Detect

Figure 3: YOLOv8 Architecture

The modification was made to the Neck component. Specifically, in the C2f block, the number of bottleneck blocks (n) was originally set to 3 but was reduced to 2. This reduction effectively decreased the depth of the network, improving computational efficiency and reducing training time without significant loss of accuracy.

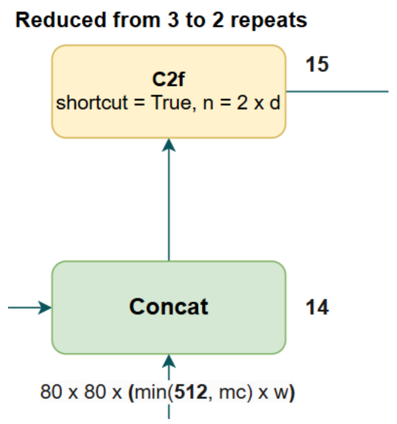
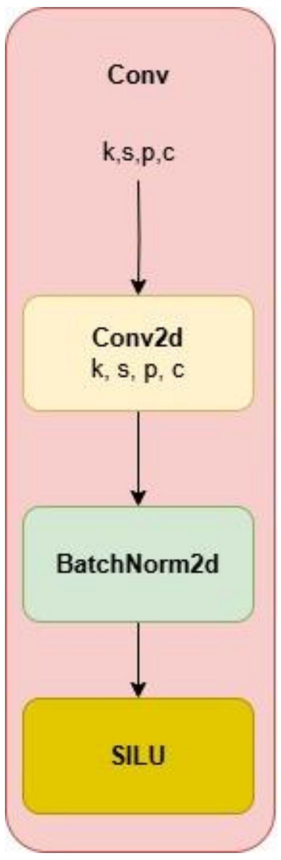
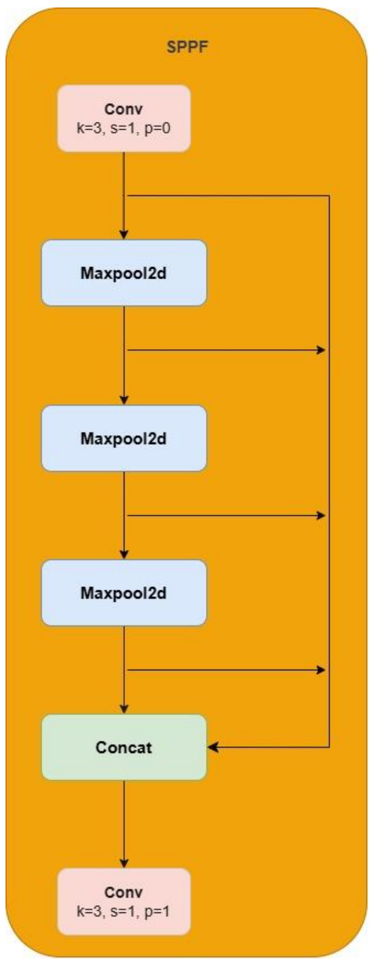

Reduced from 3 to 2 repeats
C2f
shortcut = True, n = 2 x d
15
Concat
14
80 x 80 x (min(512, mc) x w)

Figure 4: Change in C2f block

Design/Framework

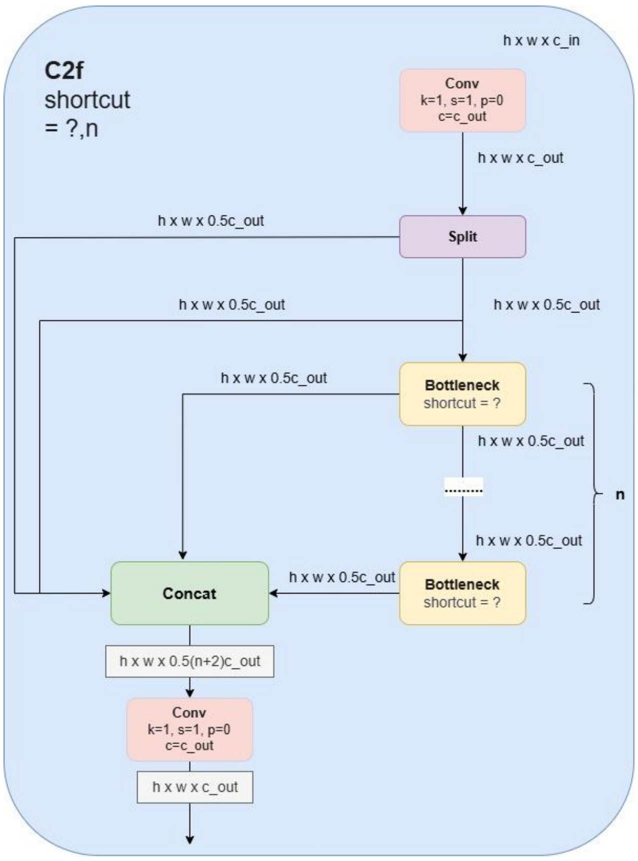
The most commonly used block in the architecture is the Convolution Block [**a**]. It consists of a 2D Convolution layer, 2D batch normalization, and SILU activation function. They are all fused as a single Convolution Block. The C2f block [b] has 2 Convolution Blocks and can have many Bottleneck Blocks.

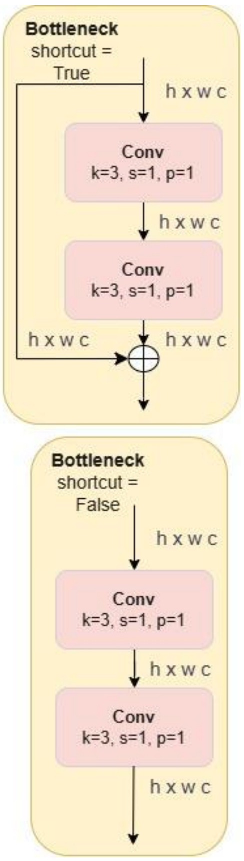




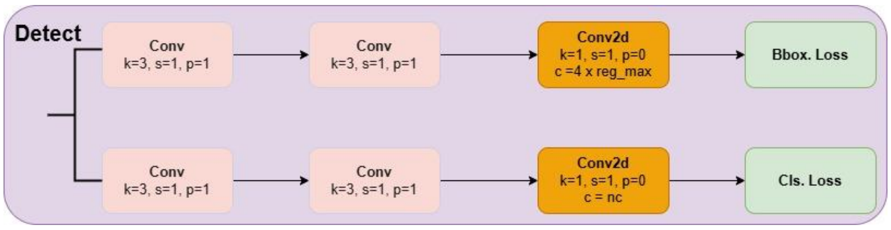
(a) Convolution Block (d) SPPF Block

Figure 5: Convolution Block & SPPF Block





(b) C2f Block (c) Bottleneck Blocks



(e) Detect Block

Figure 6: C2f Block, Bottleneck Blocks & Detect Block

Bottleneck Blocks [c] itself is a series of Conv Blocks. The SPPF Block [d] stands for Spatial Pyramid Pooling Fast. It starts with a Convolution block followed by 3 2D Max Pooling layers. Then every resulting feature map is concated right before the end of SPPF. It ends with a Convolution block. The last block is the Detect block [e]. This is where the detection happens. It contains 2 tracks. The 1st track is for bonding box prediction and the other is for class prediction. Both tracks have the same block sequence.

The Backbone is the Deep learning architecture that acts as a feature extractor. The Neck combines the features acquired from the various layers of the Backbone model. The Head predicts the classes and bonding box regions which is the final output produced by the object-detecting model.

The model has 3 parameters: Depth Multiplier (d), Width Multiplier (w) & Max Channel (mc). The values depend on the Yolo model variant (n, s, m, l, xl). Depth Multiplier (d) determines how many bottleneck blocks are in the C2f block. The Width Multiplier (w) & Max Channel (mc) determine the output channel. The YoloV8's input with 3 channels. The numbering on the flowchart [**Figure 3**] is based on the architecture file which is "yolo.yaml". It starts from the backbone, from 0, and goes until the last C2f block.

Backbone: It can be seen that the backbone is made up of numerous convolution layers, which extracts distinct features at various resolution levels. It begins with two convolutional blocks (Conv) with kernel(k) size 3 stride(s) size 2 and padding(p) 1. The spatial resolution of the output is reduced when stride(s) is 2. For example, if the input resolution in the first convolution of the block is 640x640 the output resolution after processing will be 320x320. The C2f block contains 2 parameters: shortcut and n. If the shortcut parameters in the block is "True" it indicates that the shortcut will be used on the bottleneck block whereas n determines how many bottleneck blocks will be used. The last C2f block is connected to the SPPF block. SPPF is used to generate a fixed spatial Representation of objects of various sizes, in an image without resizing the image or introducing spatial information loss

Neck: This part starts with an Upsample layer. This layer increases the SPPF's feature map resolution to match that of the C2f block. The Concat layer sums channels where the resolution is not changed.

Head: It contains 3 Detection blocks. The Detection block gets input from the neck part. The first Detection block is specialized for detecting small objects, the second one for detecting medium objects, and the third for detecting large objects.

Experiment Setup

The training and evaluation of the proposed YOLOv8 model were conducted using Google Colab, leveraging the NVIDIA T4 GPU available in the free-tier environment. The development process was carried out in Python using PyTorch and the Ultralytics YOLOv8 framework, with additional tools like OpenCV for image processing. The final model was tested and integrated into a web application using PyCharm as the primary development environment.

A custom dataset was used for training, consisting of images with annotated bounding boxes. The training process ran for 100 epochs. The loss function comprised bounding box regression loss and classification loss to ensure precise object detection. Throughout training, performance metrics such as mean average precision (mAP), precision, and recall were monitored to evaluate the model's effectiveness.

After training, the model was exported and integrated into a Flask-based web application to enable real-time object detection. The web interface allowed users to upload images and receive predictions with bounding box annotations. PyCharm was used for further debugging and refinement, ensuring the system's reliability and efficiency. This experimental setup effectively balanced computational constraints while achieving a robust object detection model suitable for deployment.

.

Summary

This section outlined the methodology followed to develop an optimized YOLOv8-based model. The modifications made to the Neck component, specifically reducing the number of bottleneck blocks, allowed for a more efficient model suitable for limited computational resources. The model formulation was detailed, showing how features are extracted, aggregated, and used for final detection. The experimental setup was described, highlighting the use of Google Colab’s free-tier GPU and the framework chosen for model development. Finally, the trained model was successfully integrated into a Flask-based web application, demonstrating its practical usability for real-time detection.

Chapter 4

Results and Discussion

Obtained Results

The proposed model was successfully trained and evaluated, achieving significant improvements in efficiency while maintaining high detection accuracy. The model was trained with a custom dataset with annotated bounding boxes, using an image size of 640 × 640**,** for 100 epochs. The trained model achieved a mean Average Precision (mAP@0.5) of 86%**,** with a mAP@0.5:0.95 of 62%**,** indicating strong detection performance across various Intersection over Union (IoU) thresholds. The model exhibited a Precision of 82% and a Recall of 78%**.**

A comparative analysis was conducted between the modified model (with a reduced number of C2f bottleneck layers) and the standard YOLOv8-s model. The modified model demonstrated a training time reduction in each epoch by 1 to 10 seconds, making it more efficient for limited-resource environments like Google Colab. Despite this reduction in depth, the mAP score only decreased by 4%. It means that the architectural change did not significantly impact performance.

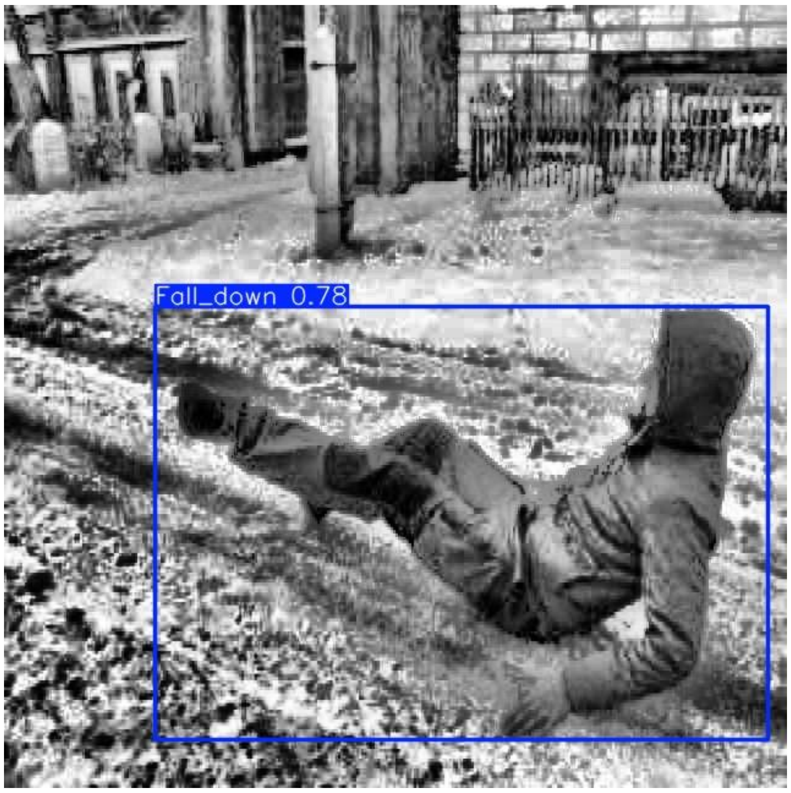


Figure 7: Test Result 01

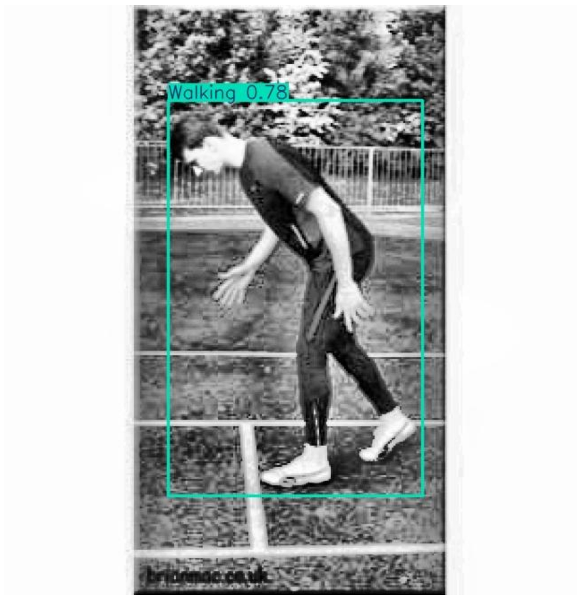


Figure 8: Test Result 02

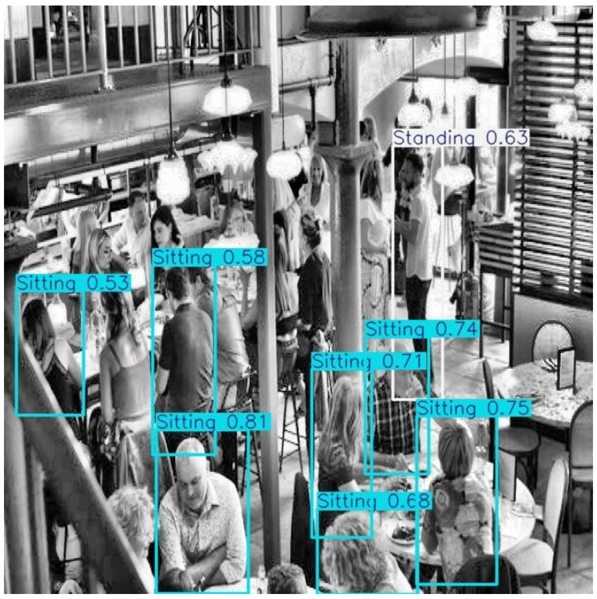


Figure 9: Test Result 03

In-Depth Result Analysis

After training the validation result,

Table 4: Validation Result

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
All	1910	2789	0.822	0.779	0.856	0.616
Fall_down	438	449	0.741	0.931	0.903	0.655
Sitting	463	877	0.888	0.707	0.846	0.592
Standing	568	716	0.843	0.726	0.819	0.583
Walking	489	747	0.817	0.751	0.854	0.634

mAP@50: Measures the model's accuracy at a 50% intersection-over-union (IoU) threshold.

mAP@50-95: A stricter metric that evaluates precision across IoU thresholds from 50% to 95%.

Precision (Box P): Indicates how many of the detected objects were correctly identified.

$$Precision = \frac{TP}{TP + FP} [10]$$

Here,

TP (True Positives): The number of correctly predicted positive instances.

FP (False Positives): The number of instances incorrectly predicted as positive.

Recall (R): Represents how many actual objects were successfully detected.

$$Recall = \frac{TP}{TP + FN} [11]$$

Here,

TP (True Positives): The number of correctly predicted positive instances.

FN (False Negatives): The number of actual positive instances that were incorrectly classified as negative.

In the Table the fall detection accuracy is the highest, with a recall of 93.1% and mAP@50 of 90.3%. The sitting and standing classes had slightly lower recall scores. The overall model accuracy is strong, with a mAP@50 of 85.6%.

Training Graph

The graph shows the training performance visualization for the model. It consists of eight subplots, each depicting training and validation metrics over the course of 100 epochs.

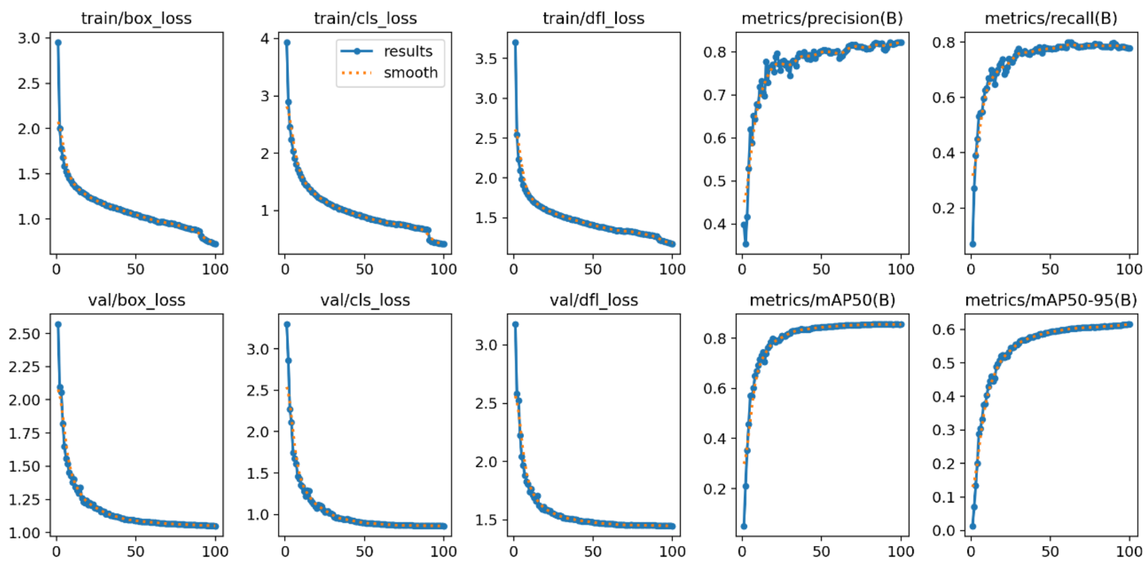


Figure 10: Training Graph

Training Loss Analysis

The plots illustrate how well the model is learning over time by monitoring different types of losses during training.

1) train/box_loss

This represents the bounding box regression loss, which measures how well the model predicts object locations. The loss starts at a high value (~3.5) and steadily decreases over the epochs, meaning that the model is learning to generate more accurate bounding boxes. The curve smooths out towards the end, indicating that the model has reached a stable training phase.

2) train/cls_loss

This plot shows the classification loss, which evaluates how well the model differentiates between classes (e.g., "falling," "sitting," "standing," etc.). The classification loss decreases rapidly in the first 20 epochs, suggesting a strong learning phase. It continues to decline but at a slower rate, indicating that the model is refining its classification ability.

3) train/dfl_loss

Distribution Focal Loss (DFL) is used to refine bounding box regression by improving localization precision. Similar to box loss, it starts at a high value (~3.8) and gradually decreases. The drop in loss means the model is enhancing object localization accuracy over training epochs.

Validation Loss Analysis

Validation loss assesses how well the model generalizes to unseen data.

1) val/box_loss
Similar to train/box_loss, but evaluated on the validation dataset. The trend is almost identical, confirming that the model is not overfitting and is generalizing well to unseen data.

2) val/cls_loss
The validation classification loss follows the same trend as the training classification loss. Since both training and validation losses decrease at a similar rate, the model is not suffering from major overfitting issues.

3) val/dfl_loss (Third from left, bottom row)
Like train/dfl_loss, this plot shows how well the bounding box refinement improves over validation data. It maintains a steady decrease, confirming that the model's localization ability is consistent across both training and validation sets.

Performance Metrics Analysis

1) metrics/precision(B)
The precision increases rapidly within the first 20 epochs and stabilizes around 85-90%. This means the model is highly accurate in predicting the correct class labels and minimizing false positives.

2) metrics/recall(B)
The recall improves quickly within the first 20 epochs, crossing 70% accuracy and stabilizing around 80%. A high recall means the model is successfully detecting most objects in the dataset.
3) metrics/mAP50(B)
The curve rises sharply within the first 30 epochs and stabilizes above 85%, confirming that the model accurately detects objects with high confidence.

4) metrics/mAP50-95(B) (Bottom-right plot)
The curve follows a similar trend as mAP50, reaching approximately 60-65%, indicating strong object detection performance across varying overlap thresholds.

Original VS Refined Dataset

The table presents a analysis of the model's performance across two different datasets: Original and Refined.

Table 5: Original VS Refined

Dataset	mAP@0.5 (%)	mAP@0.5:0.95(%)	Precision (%)	Recall (%)
Original	76	60	79	73
Refined	86	61	82	78

The Refined dataset consistently improves performance across all metrics, indicating that data augmentation and preprocessing positively impact model accuracy. The biggest improvement is in mAP@0.5 (from 76% to 86%), meaning better object localization. The mAP@0.5:0.95 increase is minimal, implying that the model still faces challenges at stricter IoU thresholds. Precision and Recall improvements indicate a better balance between false positives and false negatives, making the model more reliable.

Comparison of F1-Confidence Curves

The graphs represent F1-Score vs. Confidence Threshold curves. The F1-score is a measure of a model's accuracy, balancing precision and recall. The confidence threshold determines at what probability a prediction is considered valid.

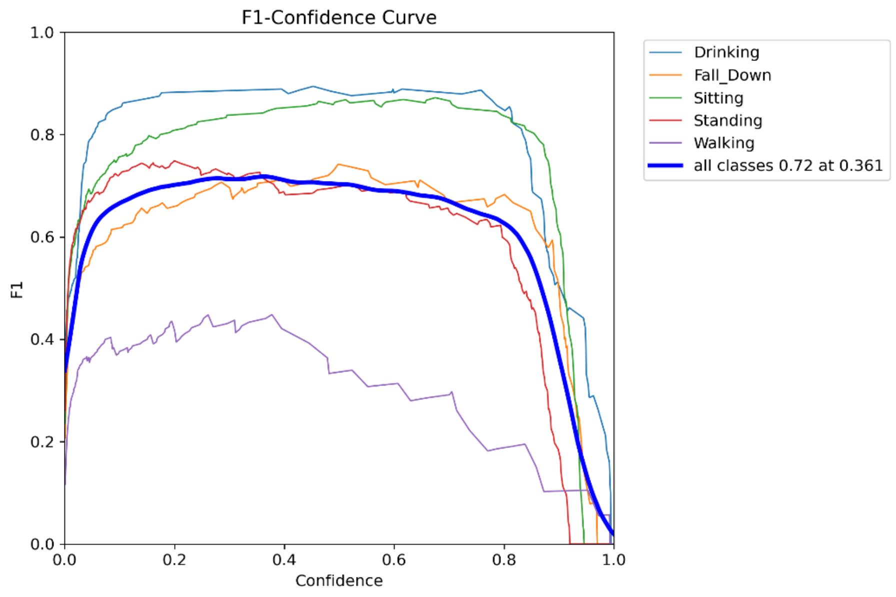


Figure 11: Old F1- Confidence Curve

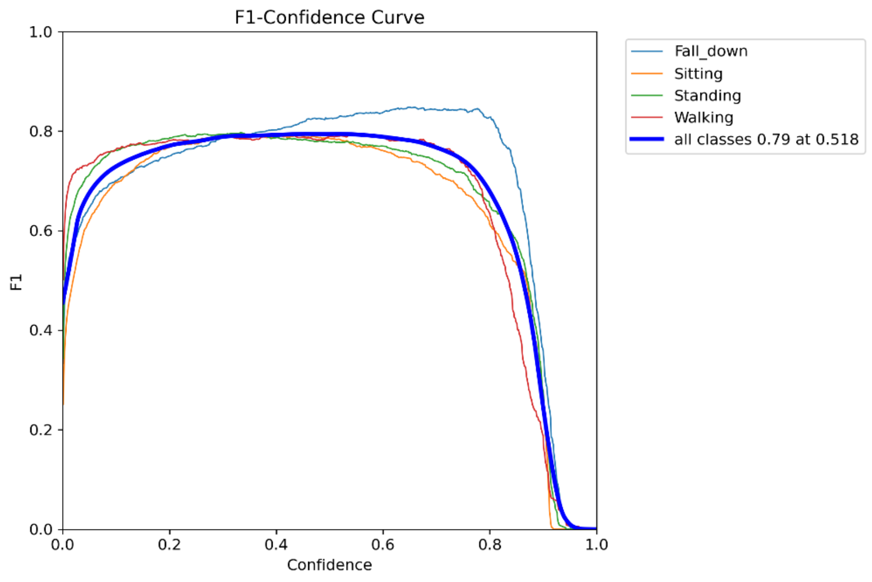


Figure 12: New F1- Confidence Curve

Table 6: Comparison of the Two F1-Confidence Curves

Aspect	First Graph (Before)	Second Graph (After)
Peak F1-Score	0.72 at 0.361	0.79 at 0.518
Confidence Threshold for Best Performance	0.361	0.518
Best Performing Class	Drinking & Sitting	Fall_Down
Worst Performing Class	Walking	Walking, but improved slightly
Overall Trend	Performance declines after 0.6 confidence	Performance remains stable until 0.8 confidence

The second model performs better overall, achieving a higher F1-score (0.79 vs. 0.72) and a more stable confidence threshold range. The Walking class still underperforms, suggesting possible dataset imbalance or feature extraction issues. Setting a higher confidence threshold (0.518 instead of 0.361) resulted in fewer false positives while maintaining a good recall.

Comparison of Precision-Recall Curves

The graphs illustrate the Precision-Recall (PR) curves. The PR curve shows the trade-off between precision (positive predictive value) and recall (true positive rate) across different confidence thresholds.

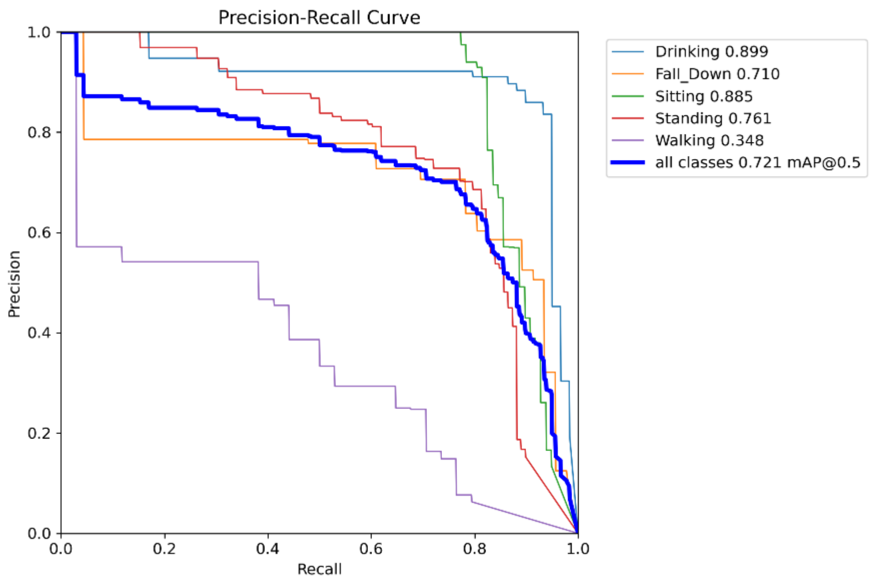


Figure 13: Old Precision-Confidence Curve

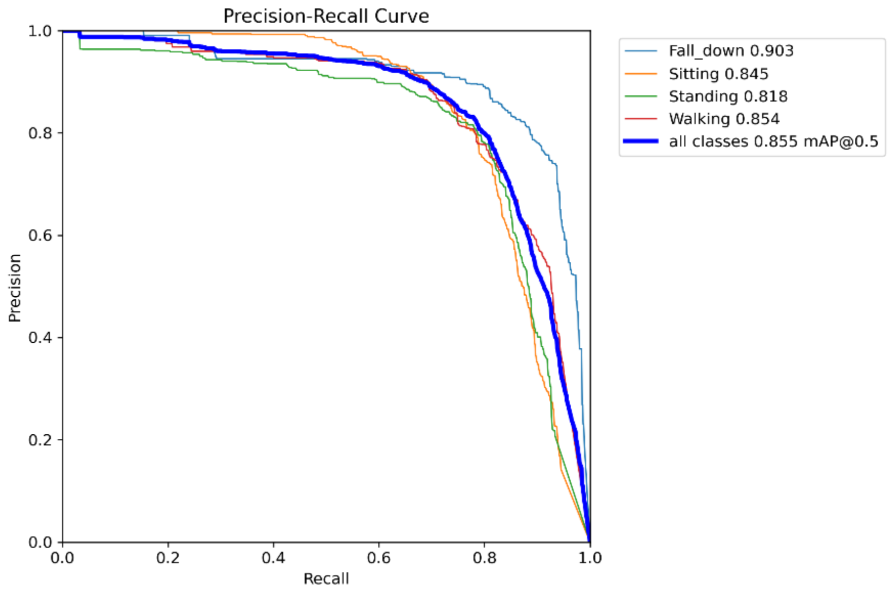


Figure 14: New Precision-Confidence Curve

Table 7: Comparison of Precision-Recall

Metric	First Graph (Before)	Second Graph (After)
mAP@0.5 (All Classes)	0.721	0.855
Best Performing Class	Drinking (0.899)	Fall_Down (0.903)
Worst Performing Class	Walking (0.348)	None (All > 0.80)
Walking Performance	0.348 (Poor)	0.854 (Good)
Overall PR Curve Stability	Inconsistent for lower classes	Balanced for all actions

After the refinement, the overall precision and recall have increased. Walking detection has the largest improvement (0.348 → 0.854). Fall_Down detection is now highly accurate (0.903),

Performance Evaluation

The following table provides a comparison between the standard YOLOv8-s model and the Modified model:

Table 8: Standard vs Modified model

Model Version	mAP@0.5 (%)	mAP@0.5:0.95 (%)	Precision (%)	Recall (%)
Standard YOLOv8-s	90	66	88	83
Modified Model	86	61	82	78

The mAP@0.5 and mAP@0.5:0.95 values indicate that the modified model maintained high detection accuracy despite the reduced depth. Precision and recall didn't show difference, proving that reducing the number of bottleneck layers did not significantly affect object detection performance.

Discussion

The results of the modified model highlight its effectiveness in balancing accuracy and computational efficiency. By reducing the number of bottleneck layers in the C2f block, the model was able to achieve a faster training time and improved inference speed, making it more practical for real-time applications. This modification is particularly valuable for scenarios where GPU resources are limited, such as Google Colab's free-tier environment.

The comparison with the standard YOLOv8-s model confirmed that the proposed modifications effectively improved computational efficiency without significantly sacrificing detection performance. The slight drop in mAP is an acceptable trade-off considering the reduction in training time. Future work could explore further optimizations, such as using quantization and pruning techniques, to further reduce model size and computational requirements while maintaining high accuracy.

Chapter 5

Conclusion

Overall Contributions

This study presented an optimized YOLOv8 model with modifications aimed at improving efficiency while maintaining high detection performance. The key contributions of this research can be summarized as follows:

Architectural Optimization for Faster Training & Inference:
A key modification was made to the Neck component of the YOLOv8-s model by reducing the number of bottleneck layers in the C2f block (n = 2 instead of 3). This resulted in a significant reduction in training time without severely impacting detection accuracy.

Enhanced Efficiency for Resource-Constrained Environments:
The modified model was designed to be computationally efficient, making it more suitable for environments with limited GPU resources, such as Google Colab's free-tier. By balancing model complexity with performance, this approach allows researchers and developers to train and deploy high-performing object detection models on low-cost hardware.

Evaluation of Real-Time Performance:
The study conducted an extensive performance analysis of the modified model, comparing it against the standard YOLOv8-s model. Metrics such as mAP, precision and recall were analyzed, confirming that the model can be used for real-time object detection applications such as video surveillance.

Detailed Comparative Analysis of Model Performance:
A class-wise performance breakdown was conducted, analyzing how the model performs across different object sizes and under various conditions. This analysis provided insights into the strengths and weaknesses of the detection pipeline, helping identify areas for future improvements.

Limitations and Future Work

Despite the successful development and optimization of the model, certain limitations were observed. These limitations highlight areas where further improvements can be made in future research.

Limitations:

1. **Detection of Small and Occluded Objects:**

While the model performed well for medium and large objects, the accuracy for small objects was slightly lower. This issue arises due to feature loss during downsampling in the Backbone and Neck components. Similarly, occluded objects were sometimes misclassified or not detected with high confidence, indicating room for improvement in context-awareness during detection.

2. **Impact of Reduced Bottleneck Layers:**

Although reducing the number of bottleneck layers in the C2f block improved efficiency, it also resulted in a slight drop in detection accuracy compared to the original YOLOv8-s model. While the trade-off between accuracy and efficiency was justified, further fine-tuning could help mitigate the accuracy drop while retaining speed improvements.

3. **Dependence on GPU Resources for Real-Time Inference:**

While the model achieved real-time performance on an NVIDIA T4 GPU, inference on CPU-based environments was significantly slower. This makes the deployment of the model challenging in edge devices or low-power embedded systems. Future work could explore model quantization and pruning techniques to enhance inference speed on low-power devices.

4. **Limited Generalization in Challenging Conditions:**

The model struggled in cases of poor lighting conditions, extreme weather scenarios, and cluttered backgrounds. These factors affected detection confidence, suggesting that additional training data augmentation strategies, such as synthetic occlusion and contrast normalization, could help improve robustness.

Future Works:

1. **Incorporating Attention Mechanisms for Improved Feature Extraction:**

Future research could explore attention-based architectures, such as Transformer-based object detection (DETR) or CBAM (Convolutional Block Attention Module), to enhance small object detection and robustness in occluded environments. These mechanisms could help retain fine details in feature maps while reducing unnecessary computations.

2. **Applying Model Compression Techniques:**

To improve inference speed on edge devices, future work could explore model quantization, pruning, and knowledge distillation. These techniques would reduce model size while maintaining detection accuracy, making the model more deployable in embedded systems, drones, and mobile applications.

3. **Extensive Hyperparameter Optimization:**

Further fine-tuning of learning rates, weight decay, and data augmentation strategies could help improve overall performance. Experimenting with different depth and width multipliers in the YOLOv8 architecture could lead to better trade-offs between accuracy and speed.

4. **Expanding the Dataset for Better Generalization:**

To improve detection in challenging conditions, additional datasets covering diverse lighting, occlusion scenarios, and extreme weather conditions should be included in training. This would help the model generalize better across real-world applications.

5. **Deploying on Edge Devices for Real-World Applications:**

Future implementations could focus on deploying the model on edge devices such as Raspberry Pi, Jetson Nano, or mobile phones. This would require optimizing the architecture for low-power hardware, making the model more practical for IoT, smart surveillance, and autonomous vehicle applications.

6. **Exploring Advanced Loss Functions for Better Training Stability:**

Implementing advanced loss functions such as focal loss (to handle class imbalances) or IoU-aware loss (to improve bounding box regression) could help enhance the model’s detection performance in difficult scenarios.

Bibliography

[1] M. Karim, S. Khalid, A. Aleryani, J. Khan, I. Ullah, and Z. Ali, “Human Action Recognition Systems: A Review of the Trends and State-of-the-Art,” *IEEE Access*, vol. 12, pp. 36372–36390, 2024, doi: 10.1109/ACCESS.2024.3373199.

[2] M. Batool *et al.*, “Multimodal Human Action Recognition Framework using an Improved CNNGRU Classifier,” *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3481631.

[3] X. Ji and H. Liu, “Advances in view-invariant human motion analysis: A review,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 40, no. 1, pp. 13–24, 2010, doi: 10.1109/TSMCC.2009.2027608.

[4] Y. Wanyan, X. Yang, W. Dong, and C. Xu, “A Comprehensive Review of Few-shot Action Recognition,” Jul. 2024.

[5] H. Kotte, M. Kravcik, and N. Duong-Trung, “Real-Time Posture Correction in Gym Exercises: A Computer Vision-Based Approach for Performance Analysis, Error Classification and Feedback,” *Proceedings of the 3rd International Workshop on Multimodal Immersive Learning Systems (MILeS’23), 18th European Conference on Technology Enhanced Learning (EC-TEL 2023)}*, vol. 3499, pp. 81–92, Oct. 2023, [Online]. Available: https://www.researchgate.net/publication/374659509_Real-Time_Posture_Correction_in_Gym_Exercises_A_Computer_Vision-Based_Approach_for_Performance_Analysis_Error_Classification_and_Feedback

[6] D. Cheng, P. Y. Chi, T. Kwak, B. Hartmann, and P. Wright, “Body-Tracking Camera Control for Demonstration Videos,” *Conference on Human Factors in Computing Systems - Proceedings*, vol. 2013-April, pp. 1185–1190, Apr. 2013, doi: 10.1145/2468356.2468568.

[7] N. Zarka, Z. Alhalah, and R. Deeb, “Real-time human motion detection and tracking,” *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA*, 2008, doi: 10.1109/ICTTA.2008.4530098.

[8] G. C. Hapsari and A. S. Prabuwono, “Human Motion Recognition in Real-time Surveillance System: A Review,” *Journal of Applied Sciences*, vol. 10, no. 22, pp. 2793–2798, 2010, doi: 10.3923/JAS.2010.2793.2798.

[9] N. Challa, “Artificial Intelligence for Object Detection and its Metadata,” vol. 2, pp. 121–133, Dec. 2023, doi: 10.17605/OSF.IO/FG3SQ.

[10] R. Madhavan and E. Messina, “Performance Metrics for Intelligent Systems (PerMIS) 2006 workshop: Summary and review,” *Proceedings - Applied Imagery Pattern Recognition Workshop*, p. 31, 2006, doi: 10.1109/AIPR.2006.29.

[11] P. Brown, S. D. Pietra, V. D. Pietra, and R. Mercer, “The Mathematics of Statistical Machine Translation: Parameter Estimation,” *International Conference on Computational Logic*, 1993.

Appendix

Program Code

Preparation

Installing necessary packages and mounting Drive

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
!pip install ultralytics
!pip install roboflow
```

Importing necessary libraries

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import yaml
from PIL import Image
import os
import cv2
import glob
from matplotlib.patches import Rectangle
import seaborn as sns
from roboflow import Roboflow
from ultralytics import YOLO
from IPython.display import display, Image
from collections import Counter
```

Import Dataset and Visualize

Downloading dataset from Roboflow

```python
rf = Roboflow(api_key=[redacted])
project = rf.workspace("activity-recognition-phelk").project("v10-rluy8")
version = project.version(1)
dataset = version.download("yolov8")
```

Setting path

```python
train_labels_folder = "/content/V10-1/train/labels"
valid_labels_folder = "/content/V10-1/valid/labels"
test_labels_folder = "/content/V10-1/test/labels"
yaml_file = "/content/V10-1/data.yaml"
```

Reading Yamal file

```
with open(yaml_file, "r") as f:
    data = yaml.safe_load(f)
class_names = data["names"]
```

Function to count instances

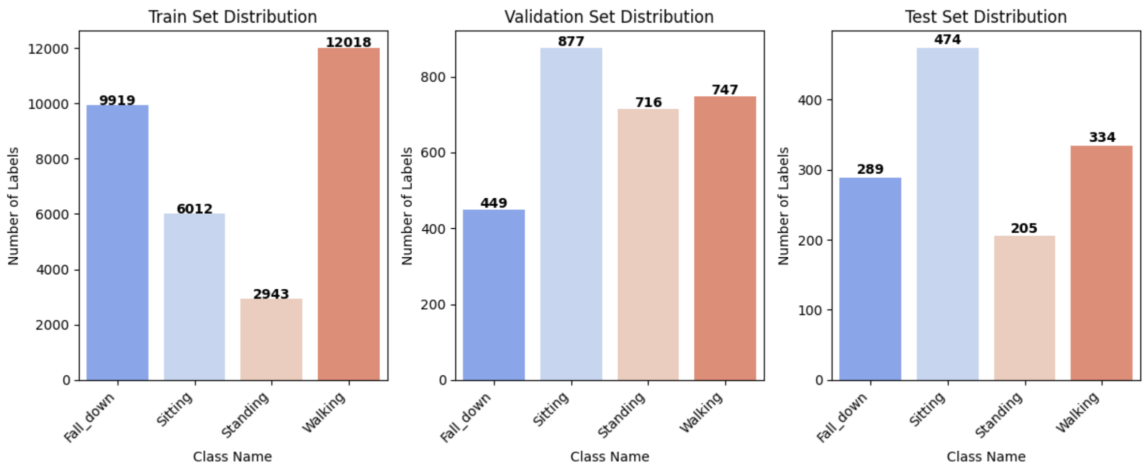
```
def count_labels(labels_folder):
    class_counts = Counter()
    label_files = [f for f in os.listdir(labels_folder) if f.endswith(".txt")]

    for label_file in label_files:
        with open(os.path.join(labels_folder, label_file), "r") as f:
            lines = f.readlines()
            for line in lines:
                class_id = int(line.split()[0])
                class_counts[class_id] += 1

    sorted_classes = sorted(class_counts.keys())
    sorted_counts = [class_counts[c] for c in sorted_classes]
    sorted_labels = [class_names[c] for c in sorted_classes]

    return sorted_labels, sorted_counts
```

Class distribution

```
train_labels, train_percentages = count_labels(train_labels_folder)
valid_labels, valid_percentages = count_labels(valid_labels_folder)
test_labels, test_percentages = count_labels(test_labels_folder)
```

Plot bar chart

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Train Plot
sns.barplot(x=train_labels, y=train_counts, hue=train_labels, palette="coolwarm", ax=axes[0], legend=False)
axes[0].set_title("Train Set Distribution")
axes[0].set_xlabel("Class Name")
axes[0].set_ylabel("Number of Labels")
axes[0].set_xticks(range(len(train_labels)))
axes[0].set_xticklabels(train_labels, rotation=45, ha="right")
for i, v in enumerate(train_counts):
    axes[0].text(i, v + 5, f"{v}", ha="center", fontsize=10, fontweight="bold")

# Validation Plot
sns.barplot(x=valid_labels, y=valid_counts, hue=valid_labels, palette="coolwarm", ax=axes[1], legend=False)
axes[1].set_title("Validation Set Distribution")
axes[1].set_xlabel("Class Name")
axes[1].set_ylabel("Number of Labels")
axes[1].set_xticks(range(len(valid_labels)))
axes[1].set_xticklabels(valid_labels, rotation=45, ha="right")
for i, v in enumerate(valid_counts):
    axes[1].text(i, v + 5, f"{v}", ha="center", fontsize=10, fontweight="bold")

# Test Plot
sns.barplot(x=test_labels, y=test_counts, hue=test_labels, palette="coolwarm", ax=axes[2], legend=False)
axes[2].set_title("Test Set Distribution")
axes[2].set_xlabel("Class Name")
axes[2].set_ylabel("Number of Labels")
axes[2].set_xticks(range(len(test_labels)))
axes[2].set_xticklabels(test_labels, rotation=45, ha="right")
for i, v in enumerate(test_counts):
    axes[2].text(i, v + 5, f"{v}", ha="center", fontsize=10, fontweight="bold")

plt.tight_layout()
plt.show()
```



Function to count total instances

```
def count_total_labels(labels_folder):
    total_labels = 0
    if not os.path.exists(labels_folder):
        print(f"Error: Path not found - {labels_folder}")
        return 0

    label_files = [f for f in os.listdir(labels_folder) if f.endswith(".txt")]

    for label_file in label_files:
        with open(os.path.join(labels_folder, label_file), "r") as f:
            lines = f.readlines()
            total_labels += len(lines)

    return total_labels
```

Print total instances

```
train_total_labels = count_total_labels(train_labels_folder)
valid_total_labels = count_total_labels(valid_labels_folder)
test_total_labels = count_total_labels(test_labels_folder)

print(f"Total Instances in Train Set: {train_total_labels}")
print(f"Total Instances in Validation Set: {valid_total_labels}")
print(f"Total Instances in Test Set: {test_total_labels}")
```

```
Total Instances in Train Set: 30892
Total Instances in Validation Set: 2789
Total Instances in Test Set: 1302
```

Training Model

Modifying Yaml file

Setting class number to 4. In Head section reducing bottleneck layer of C2f block by 1.

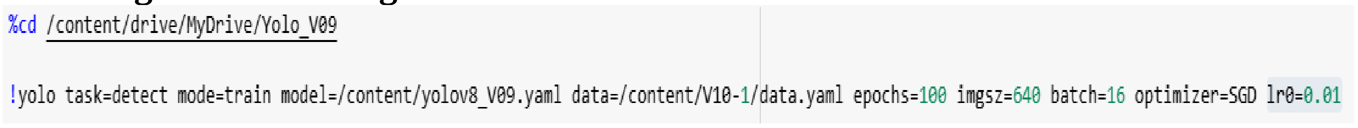
```
# Parameters
nc: 80 # number of classes
scales:
  # [depth, width, max_channels]
  s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers,
11166560 parameters, 11166544 gradients,  28.8 GFLOPs

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2f, [512]] # 12

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 3, C2f, [256]] # 15 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 3, C2f, [512]] # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]] # cat head P5
  - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

Orginal

```
# Parameters
nc: 4  # number of classes
scales:
  s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers,
11166560 parameters, 11166544 gradients,  28.8 GFLOPs

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 2, C2f, [512]] #12 Reduced from 3 to 2 repeats

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 2, C2f, [256]] #15 (P3/8-small) Reduced from 3
to 2 repeats

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 2, C2f, [512]] #18 (P4/16-medium) Reduced from 3
to 2 repeats

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]] # cat head P5
  - [-1, 2, C2f, [1024]] # 21 (P5/32-large) Reduced from
3 to 2 repeats

  - [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

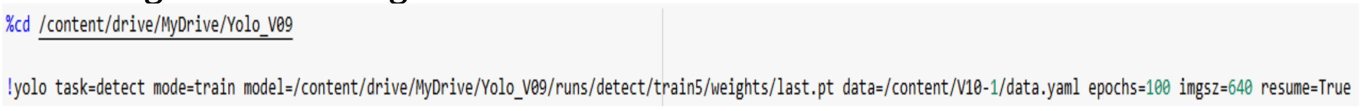
Modified

Initiating model training



```
%cd /content/drive/MyDrive/Yolo_V09

!yolo task=detect mode=train model=/content/yolov8_V09.yaml data=/content/V10-1/data.yaml epochs=100 imgsz=640 batch=16 optimizer=SGD lr0=0.01
```

Resuming model training



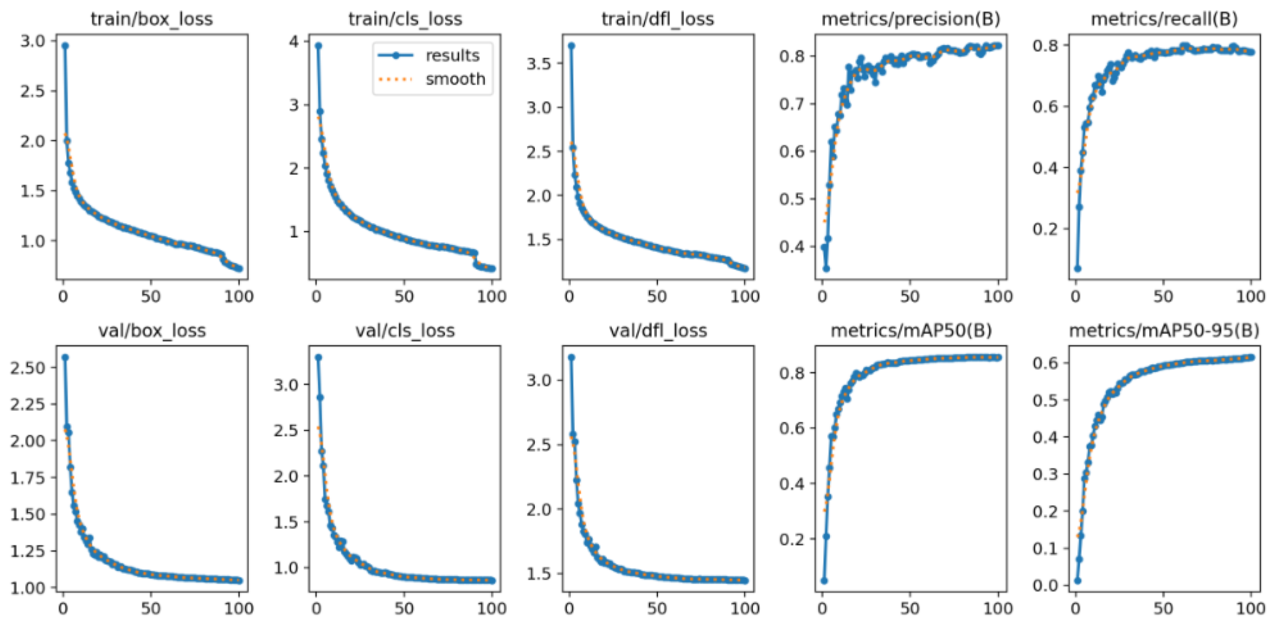
```
%cd /content/drive/MyDrive/Yolo_V09

!yolo task=detect mode=train model=/content/drive/MyDrive/Yolo_V09/runs/detect/train5/weights/last.pt data=/content/V10-1/data.yaml epochs=100 imgsz=640 resume=True
```

Training Result

```
    Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
      all       1910       2789      0.822      0.778      0.855      0.615
Fall_down        438        449       0.74      0.931      0.903      0.655
  Sitting        463        877      0.887      0.706      0.845      0.592
 Standing        568        716      0.843      0.726      0.818      0.581
  Walking        489        747      0.818      0.751      0.854      0.633
```

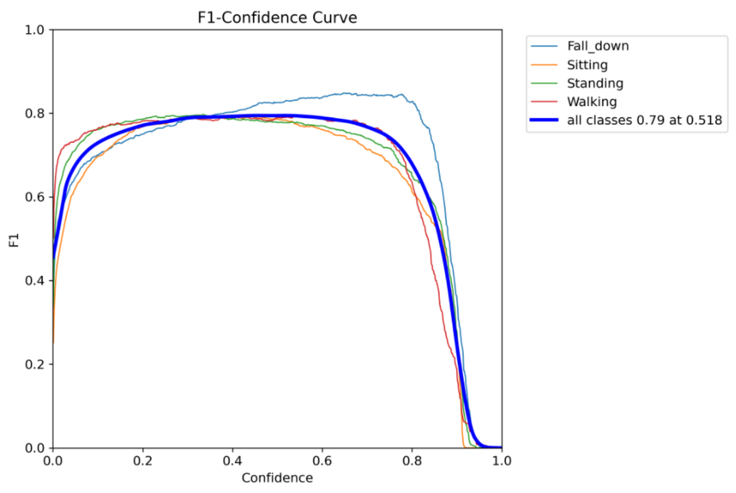
Training Graph

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/results.png", width=1000)
```



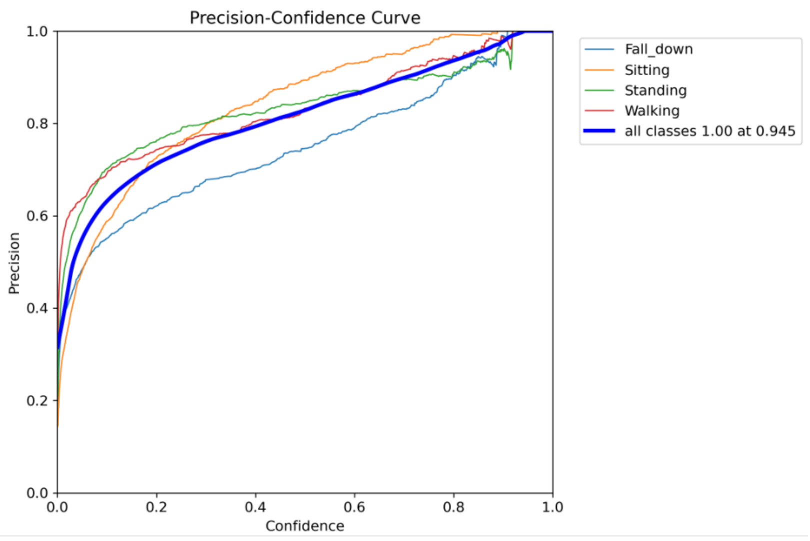
F1-Confidence Curve

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/F1_curve.png", width=1000)
```



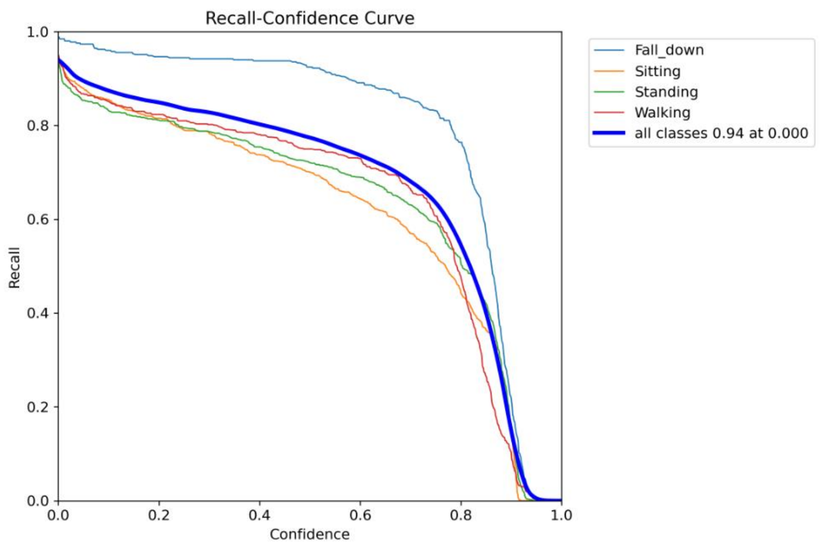
Precision-Confidence Curve

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/P_curve.png", width=1000)
```



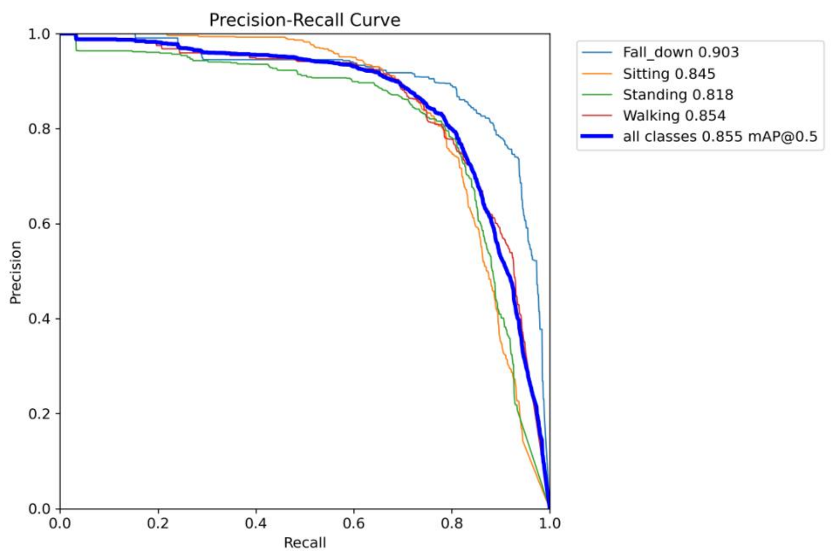
Recall-Confidence Curve

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/R_curve.png", width=1000)
```



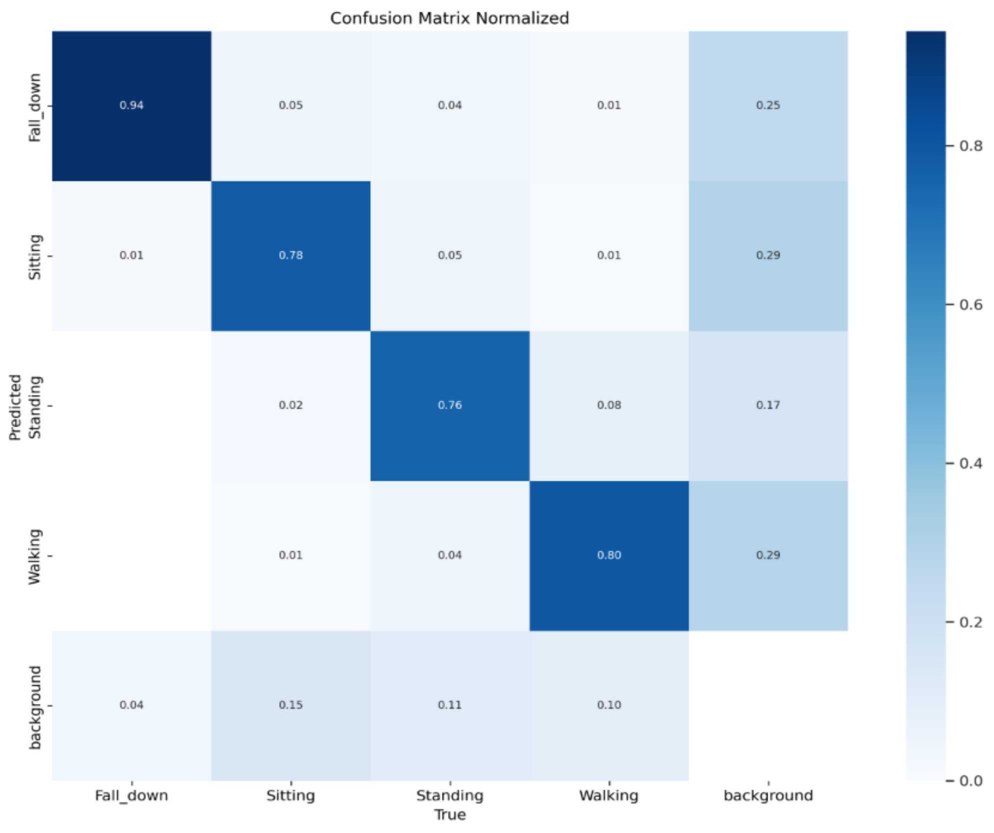
Precision-Recall Curve

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/PR_curve.png", width=1000)
```



Confusion Matrix

```
Image(filename=f"/content/drive/MyDrive/Yolo_V09/runs/detect/train5/confusion_matrix_normalized.png", width=1000)
```



	Fall_down	Sitting	Standing	Walking	background
Fall_down	0.94	0.05	0.04	0.01	0.25
Sitting	0.01	0.78	0.05	0.01	0.29
Standing		0.02	0.76	0.08	0.17
Walking		0.01	0.04	0.80	0.29
background	0.04	0.15	0.11	0.10	

Validation and Test

Validation

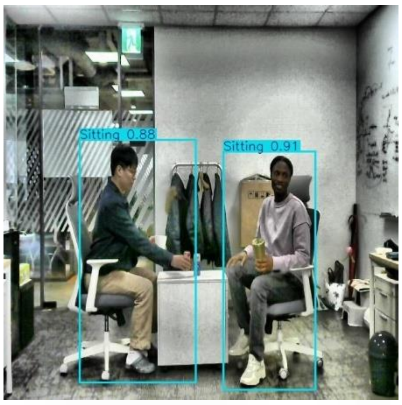
```
%cd /content/drive/MyDrive/Yolo_V09

!yolo task=detect mode=val model=/content/drive/MyDrive/Yolo_V09/runs/detect/train5/weights/best.pt data=/content/V10-1/data.yaml
```

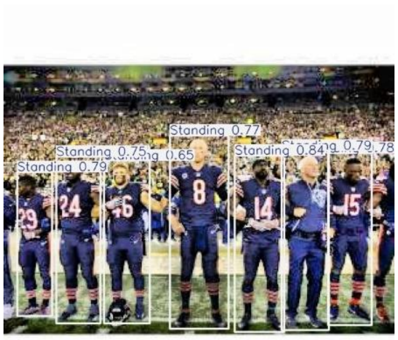
```
/content/drive/MyDrive/Yolo_V09
Ultralytics 8.3.78 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
YOLOv8_V09 summary (fused): 72 layers, 11,127,132 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /content/V10-1/valid/labels.cache... 1910 images, 2 backgrounds, 0 corrupt: 100% 1910/1910 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 120/120 [00:26<00:00,  4.46it/s]
                   all       1910       2789      0.822      0.779      0.856      0.616
             Fall_down        438        449      0.741      0.931      0.903      0.655
               Sitting        463        877      0.888      0.707      0.846      0.592
              Standing        568        716      0.843      0.726      0.819      0.583
               Walking        489        747      0.817      0.751      0.854      0.634
```

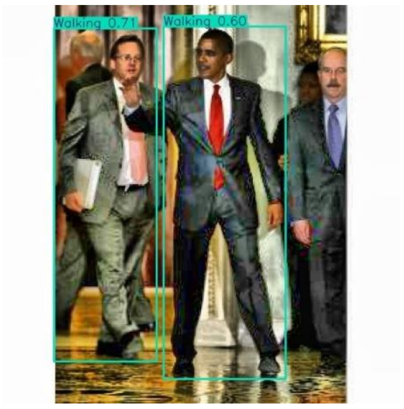
Prediction

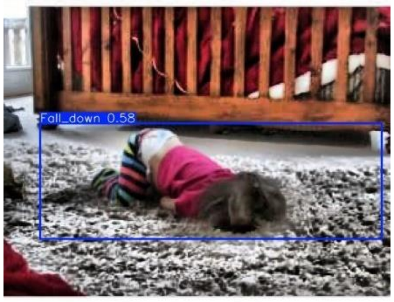
```
%cd /content/drive/MyDrive/Yolo_V09

!yolo task=detect mode=predict model=/content/drive/MyDrive/Yolo_V09/runs/detect/train5/weights/best.pt conf=0.5 source=/content/V10-1/test/images save=True
```

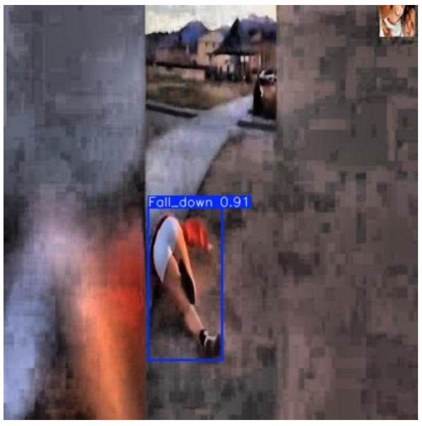
Prediction Result







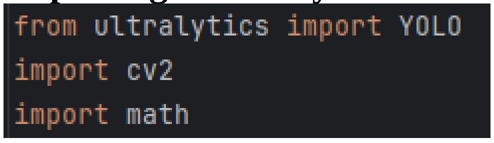




Deploying App

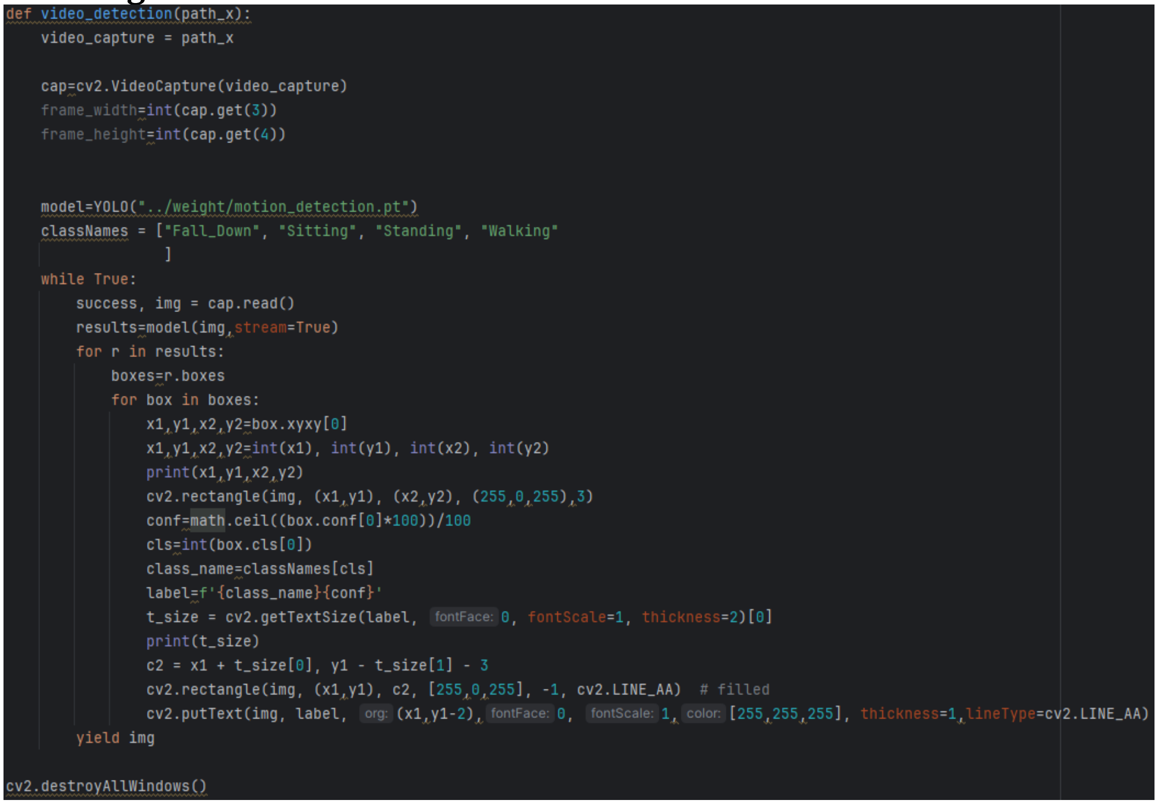
Creating Video Detection Library

Importing necessary libraries



```python
from ultralytics import YOLO
import cv2
import math
```

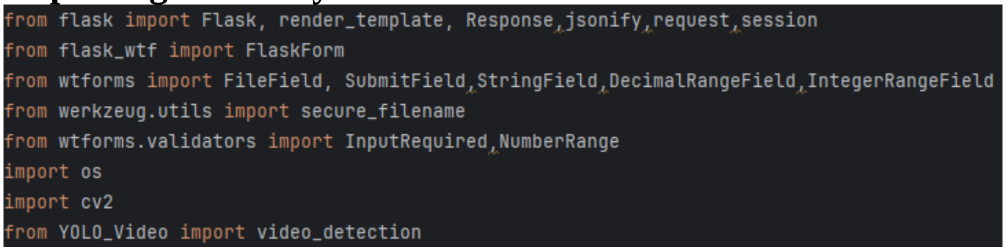
Creating video detection function



```python
def video_detection(path_x):
    video_capture = path_x

    cap=cv2.VideoCapture(video_capture)
    frame_width=int(cap.get(3))
    frame_height=int(cap.get(4))


    model=YOLO("../weight/motion_detection.pt")
    classNames = ["Fall_Down", "Sitting", "Standing", "Walking"
                  ]
    while True:
        success, img = cap.read()
        results=model(img,stream=True)
        for r in results:
            boxes=r.boxes
            for box in boxes:
                x1,y1,x2,y2=box.xyxy[0]
                x1,y1,x2,y2=int(x1), int(y1), int(x2), int(y2)
                print(x1,y1,x2,y2)
                cv2.rectangle(img, (x1,y1), (x2,y2), (255,0,255),3)
                conf=math.ceil((box.conf[0]*100))/100
                cls=int(box.cls[0])
                class_name=classNames[cls]
                label=f'{class_name}{conf}'
                t_size = cv2.getTextSize(label, fontFace: 0, fontScale=1, thickness=2)[0]
                print(t_size)
                c2 = x1 + t_size[0], y1 - t_size[1] - 3
                cv2.rectangle(img, (x1,y1), c2, [255,0,255], -1, cv2.LINE_AA)  # filled
                cv2.putText(img, label, org: (x1,y1-2), fontFace: 0, fontScale: 1, color: [255,255,255], thickness=1,lineType=cv2.LINE_AA)
        yield img

cv2.destroyAllWindows()
```

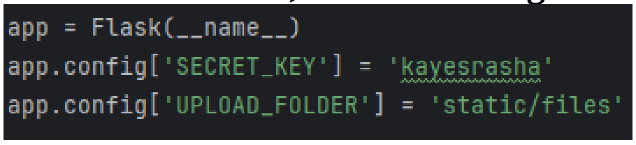
Developing Flask App

Importing necessary libraries



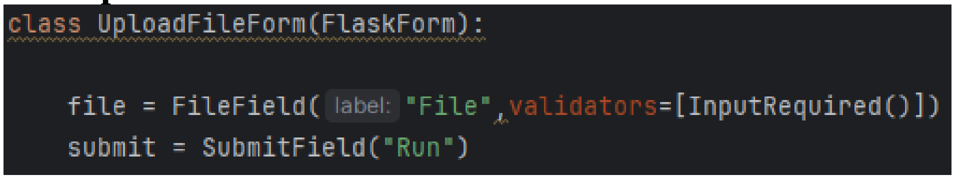
```
from flask import Flask, render_template, Response,jsonify,request,session
from flask_wtf import FlaskForm
from wtforms import FileField, SubmitField,StringField,DecimalRangeField,IntegerRangeField
from werkzeug.utils import secure_filename
from wtforms.validators import InputRequired,NumberRange
import os
import cv2
from YOLO_Video import video_detection
```

Instance Creation, Session Management, and File Upload Handling



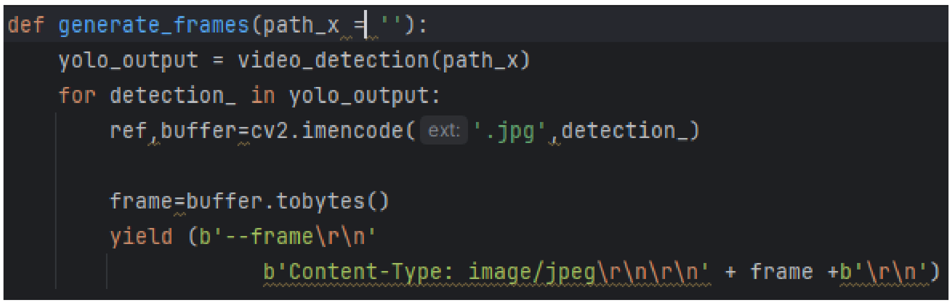
```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'kayesrasha'
app.config['UPLOAD_FOLDER'] = 'static/files'
```

File Upload Form



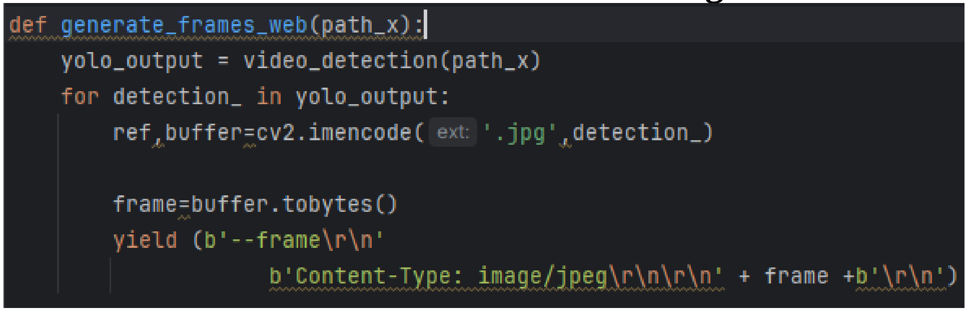
```
class UploadFileForm(FlaskForm):

    file = FileField( label: "File",validators=[InputRequired()])
    submit = SubmitField("Run")
```

Real-Time YOLOv8 Video Frame Generator



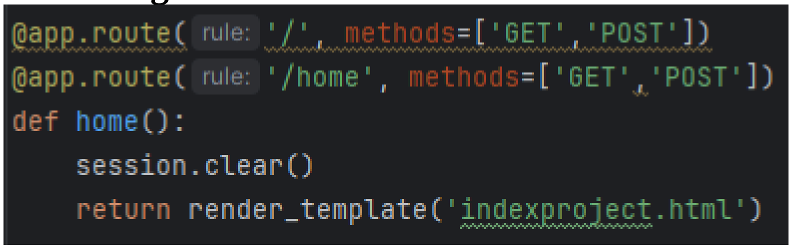
```
def generate_frames(path_x = ''):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode( ext: '.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                    b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')
```

Web-Based Frame Generation for Streaming



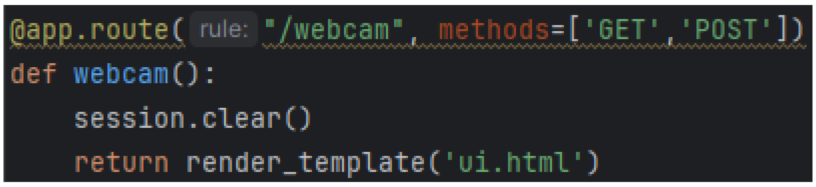
```
def generate_frames_web(path_x):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode( ext: '.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                    b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')
```

Home Page Route



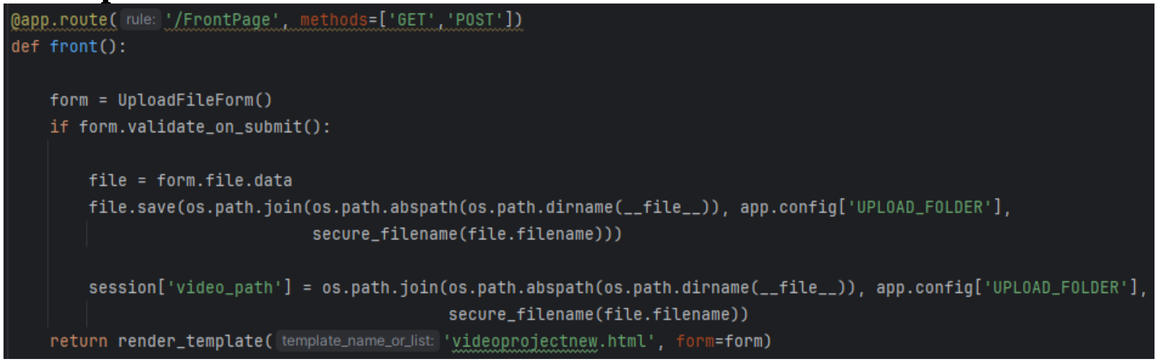
```
@app.route( rule: '/', methods=['GET','POST'])
@app.route( rule: '/home', methods=['GET','POST'])
def home():
    session.clear()
    return render_template('indexproject.html')
```

Webcam Route



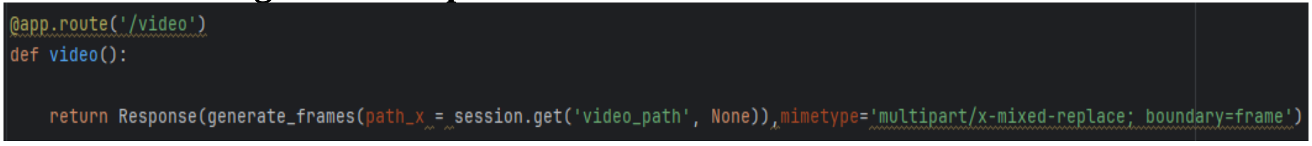
```
@app.route( rule: "/webcam", methods=['GET','POST'])
def webcam():
    session.clear()
    return render_template('ui.html')
```

File Upload Handler



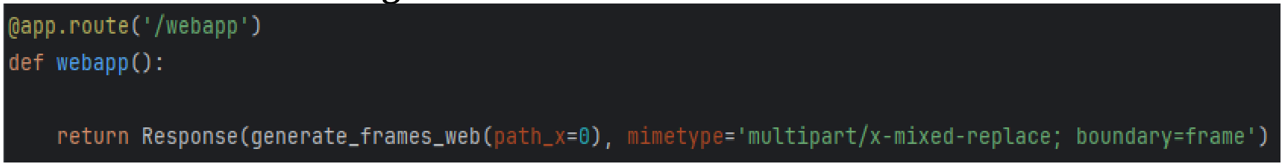
```
@app.route( rule: '/FrontPage', methods=['GET','POST'])
def front():

    form = UploadFileForm()
    if form.validate_on_submit():

        file = form.file.data
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                               secure_filename(file.filename)))

        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                                             secure_filename(file.filename))
    return render_template( template_name_or_list: 'videoprojectnew.html', form=form)
```

Video Streaming from an Uploaded File



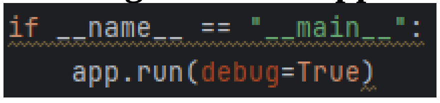
```
@app.route('/video')
def video():

    return Response(generate_frames(path_x = session.get('video_path', None)),mimetype='multipart/x-mixed-replace; boundary=frame')
```

Live Webcam Streaming



```
@app.route('/webapp')
def webapp():

    return Response(generate_frames_web(path_x=0), mimetype='multipart/x-mixed-replace; boundary=frame')
```

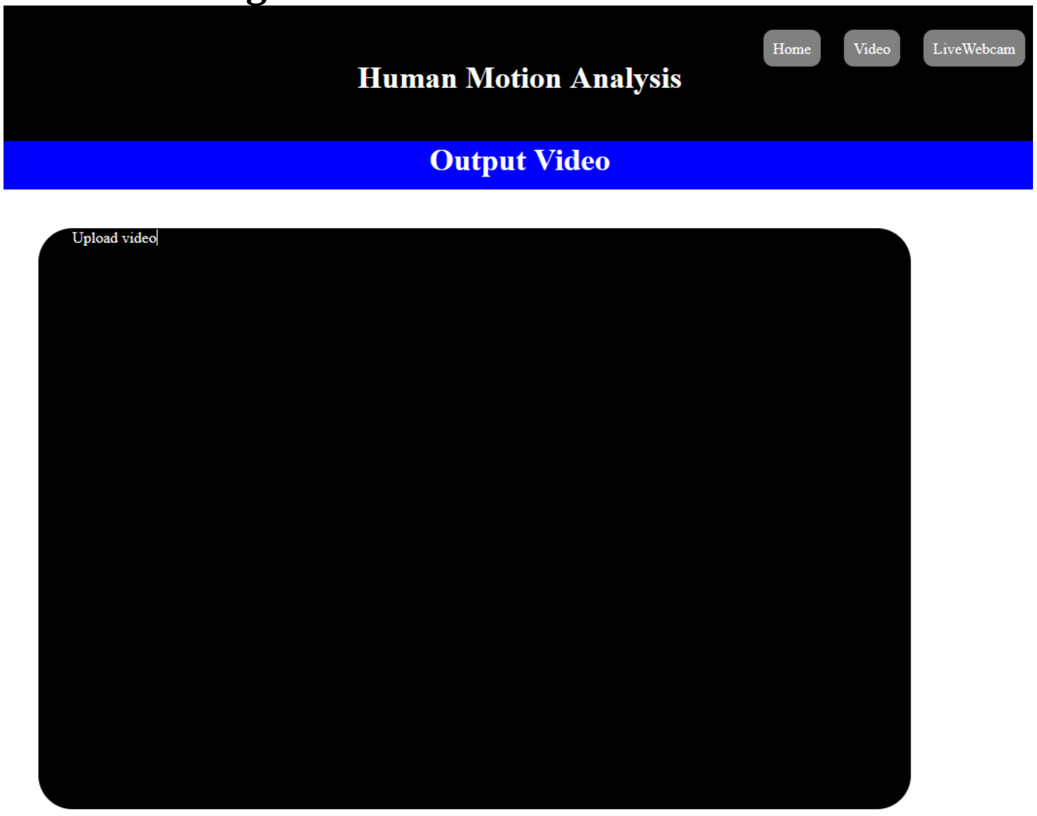
Running the Flask App



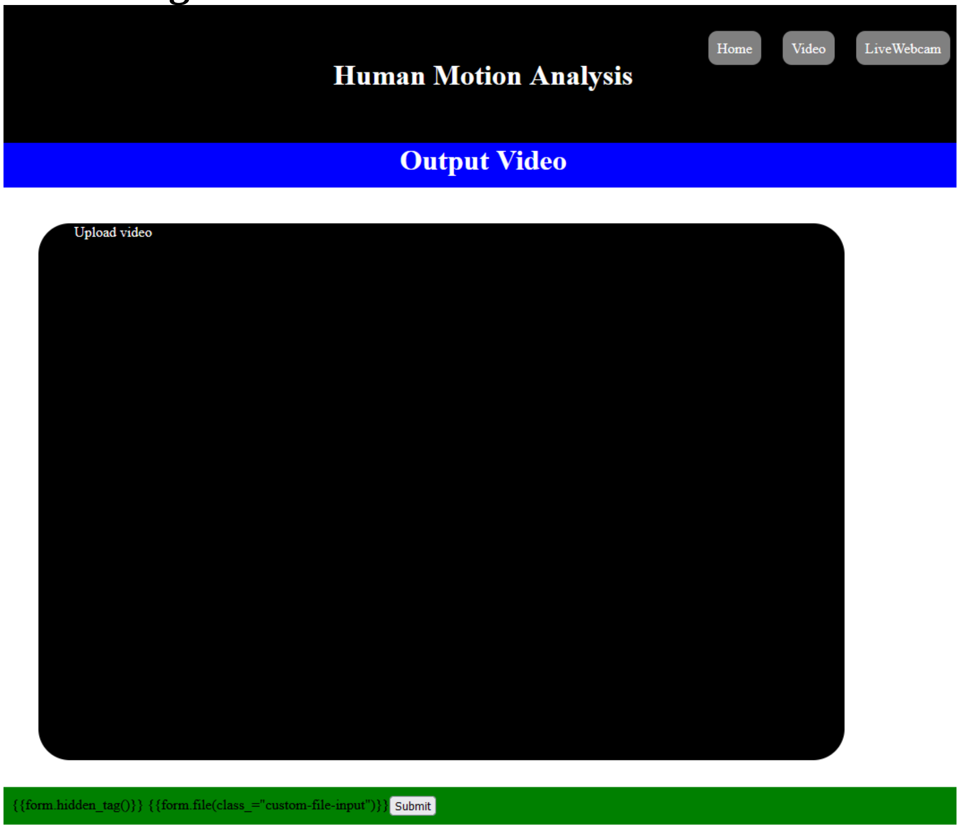
```
if __name__ == "__main__":
    app.run(debug=True)
```

Home Page

Web Cam Page



Video Page



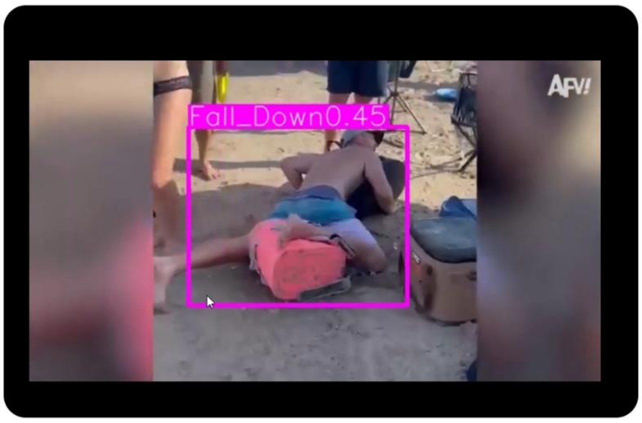
Human Motion Analysis

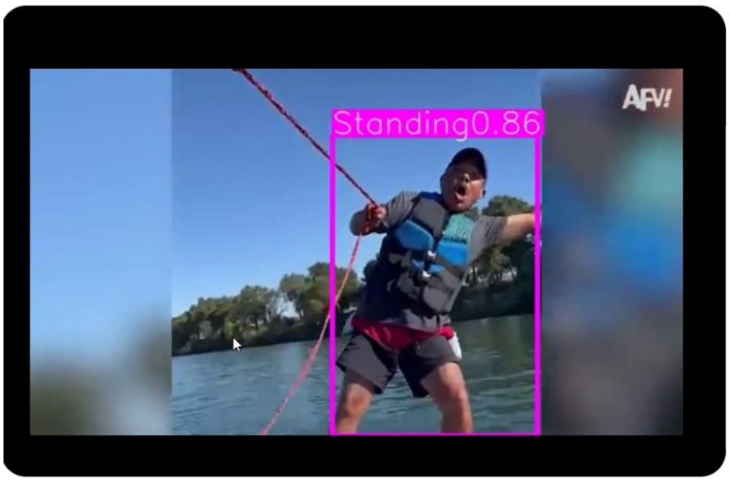
Output Video

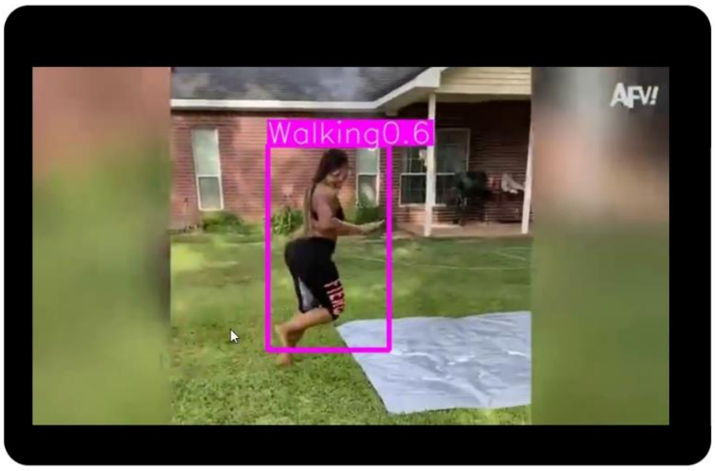
Upload video

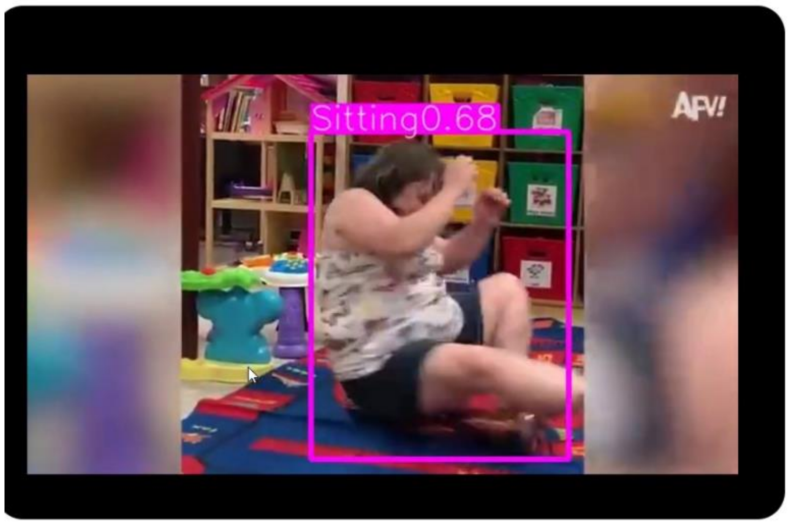
{{form.hidden_tag()}} {{form.file(class_="custom-file-input")}}

Video Result

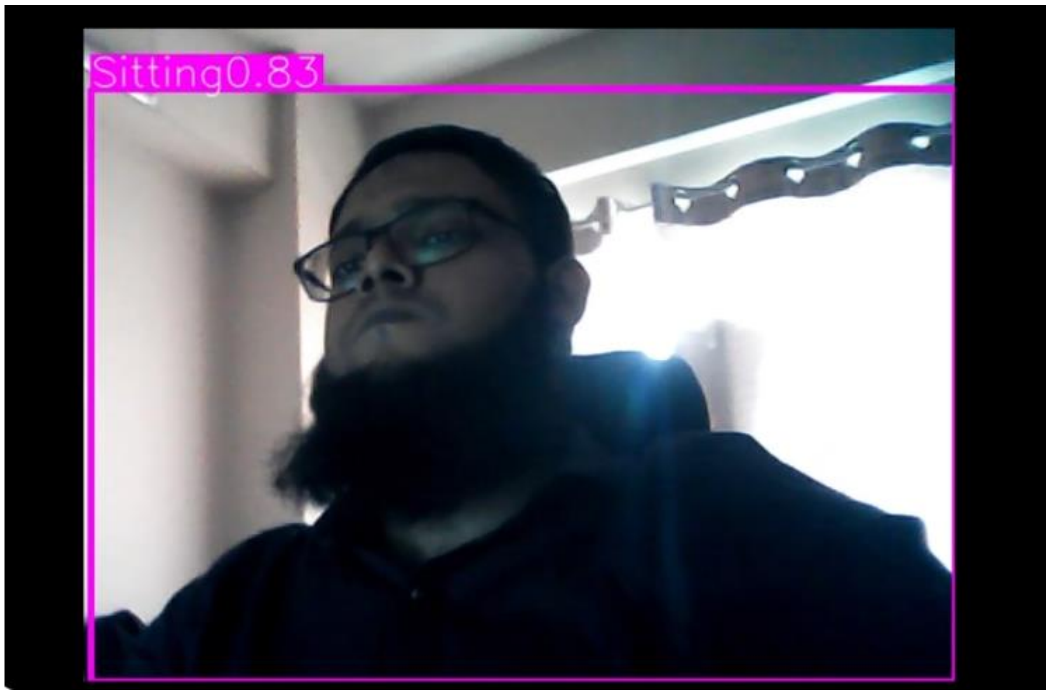


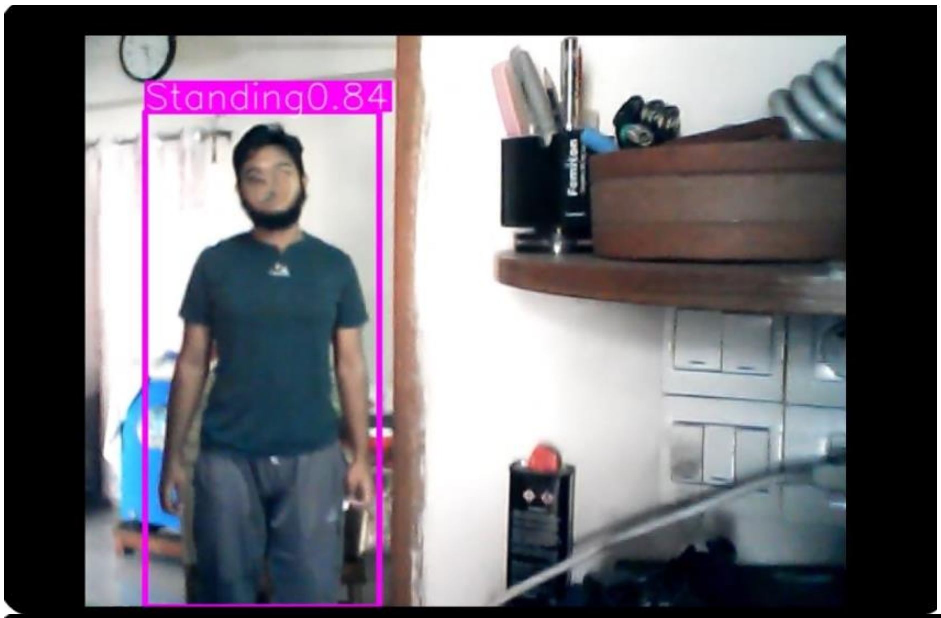


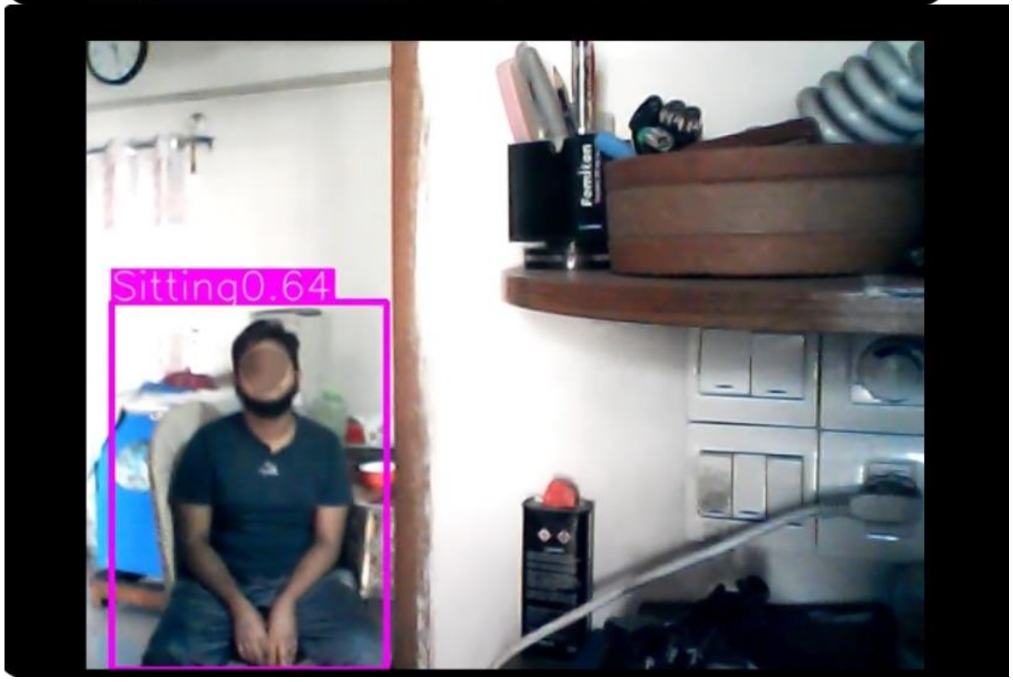




Web Cam Result




Standing0.84


Sitting0.64