**Rashtreeya Sikshana Samithi Trust**

**RV UNIVERSITY**

**School of Computer Science and Engineering**

**Bengaluru – 560 059**



**Automated PPE Detection for Workplace Safety using Object Detection and CNNs**

Submitted in partial fulfilment of the requirements for the

**UG – Research IV**

**By**

| **Vandana M** | **1RVU22BSC107** |
| --- | --- |
| **Chinmayi UH** | **1RVU22BSC022** |

**Under the guidance of**

**CVSN Reddy, Professor, SOCSE RV University**

A logo for a university

Description automatically generated

**CERTIFICATE**

*This is to certify that the Report entitled*

***Automated PPE Detection for Workplace Safety using Object Detection and CNNs***

*is a Bonafide work carried out by*

| **Vandana M** | **1RVU22BSC107** |
| --- | --- |
| **Chinmayi UH** | **1RVU22BSC022** |

In partial fulfilment for the completion of 6th semester Interdisciplinary Project (CS3710) under rules and regulations of RV University, Bengaluru during the period Jan – May 2025. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements in respect of Interdisciplinary Project work.

| **CVSN Reddy** | **Dr. Merin Thomas** | **Dr. G Shobha** |
| --- | --- | --- |
| **Professor , SoCSE** | **Assistant Dean, SoCSE** | **Dean, SoCSE** |

| **Name of the Examiner(s)** | **Signature with Date** |
| --- | --- |

**DECLARATION**

We, **Vandana M, 1RVU22BSC107 and Chinmayi UH, 1RVU22BSC022,** hereby declare that the report entitled, ***‘Automated PPE Detection for Workplace Safety using Object Detection and CNNs,*** is an original work done by us under the guidance of **CVSN Reddy, Professor, School of Computer Science and Engineering, RV University**,is being submitted in partial fulfilment of the requirements for completion of 6th semester Interdisciplinary Project.

Further, we declare that the content of the report has not been submitted previously by anybody, anywhere. We also declare that any Intellectual Property Rights generated out of this project will be the property of RV University, Bengaluru and we will be one of the authors of the same.

**PLACE:**

**DATE:**

| **Vandana M** | **1RVU22BSC107** |
| --- | --- |
| **Chinmayi UH** | **1RVU22BSC022** |

# 1. Introduction

## 1.1 Project Domain and Problem Addressing

This project belongs to the domain of **industrial safety compliance monitoring**, focusing on the automated detection of Personal Protective Equipment (PPE) usage at industrial sites. It addresses the limitations of manual PPE compliance checks, which are labour-intensive, error-prone, and inefficient, especially in large or hazardous environments. By leveraging deep learning—specifically the YOLOv9-e object detection model—this project automates the detection of essential safety gear such as **helmets**, **vests**, and **gloves** from images or video footage in real time. The goal is to improve workplace safety, reduce human error, and ensure adherence to safety protocols in dynamic industrial settings.

The increasing frequency of workplace accidents, particularly in high-risk sectors such as construction, mining, and manufacturing, underscores the need for robust safety monitoring systems. Personal Protective Equipment (PPE) plays a critical role in minimizing injuries and fatalities. Despite regulations, compliance remains a challenge due to the inefficiency of manual monitoring. Our project offers a deep learning-based solution using real-time object detection to automate the identification of PPE items on workers. This system is non-invasive, scalable, and highly accurate, leveraging the latest in CNNs and real-time detection frameworks.

## 1.2 Issues and Challenges

This project tackles several significant challenges in PPE compliance detection:

* **Manual Inspection Limitations**: Human supervisors cannot monitor every area of an industrial site consistently, leading to potential safety oversights.

* **Real-time Monitoring Needs**: Traditional systems lack the speed and accuracy required for live surveillance, resulting in delayed responses to violations.

* **Class Similarity and Misclassification**: Differentiating between PPE components like gloves and background elements can be difficult due to overlapping features.

* **Data Diversity**: Acquiring diverse, high-quality datasets that include various angles, lighting conditions, and PPE types is essential yet challenging.

* **Hardware Constraints**: Deploying such models on edge devices requires optimization for speed and low latency.

## 1.3 Problem Statement

Ensuring the proper use of PPE at industrial sites is essential for preventing workplace injuries and complying with safety regulations. However, current monitoring systems are either manual or limited in scope, leading to inconsistent enforcement. The problem addressed by this project is the **lack of an automated, scalable, and real-time system** that can detect whether workers are wearing mandatory PPE such as helmets, vests, and gloves. The aim is to reduce dependence on manual surveillance, ensure continuous monitoring, and increase compliance through an intelligent system powered by deep learning. Manual monitoring of PPE compliance is labour-intensive, subjective, and often unreliable. This can lead to serious workplace accidents and legal consequences for companies. The problem is to develop a real-time, AI-based system capable of detecting and verifying the proper usage of various PPE components using object detection models trained on industrial datasets.

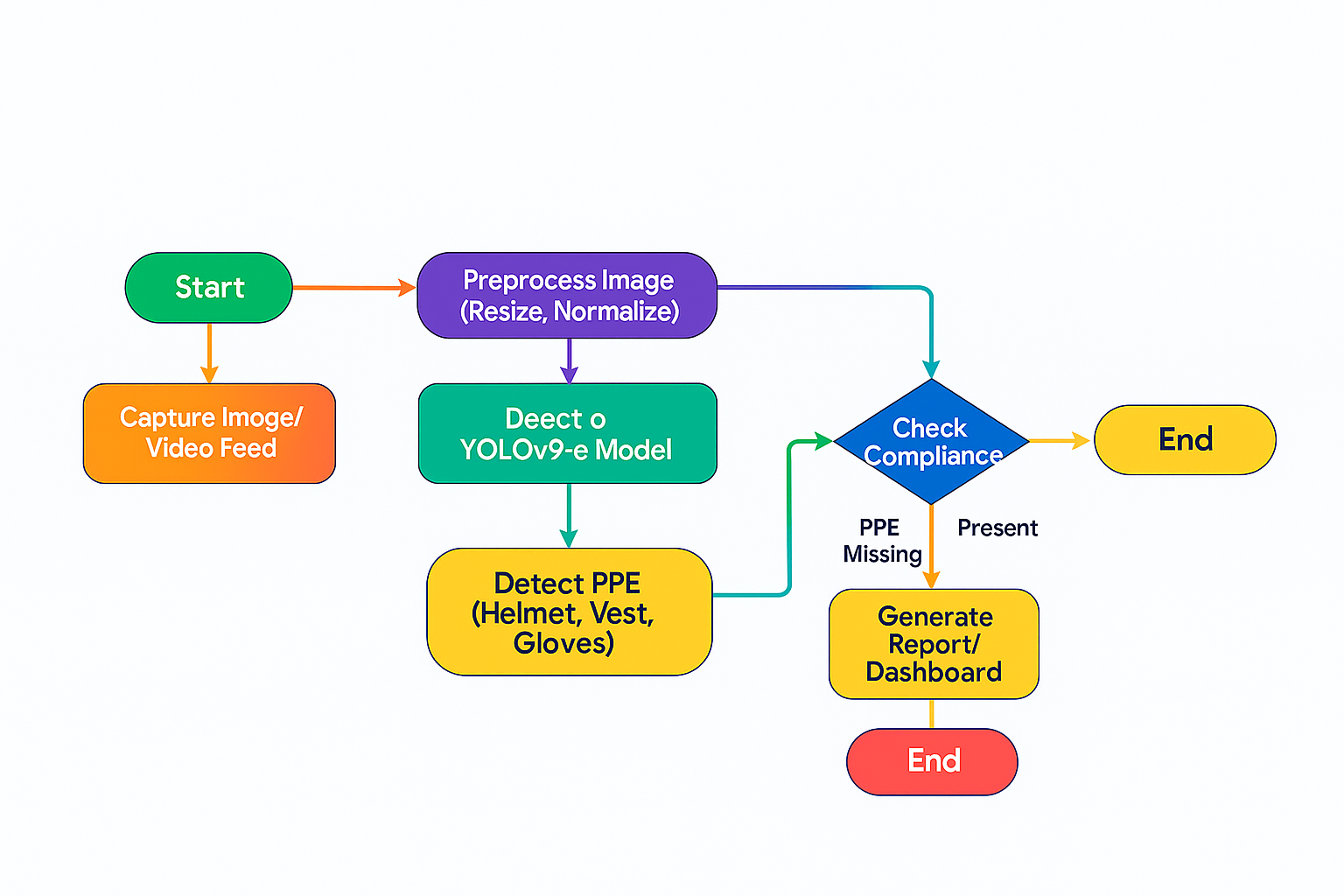


Figure 1: architecture diagram on how to approach the project

## 1.4 Requirements and Feasibility Study

### 1. Data Requirements

The model requires a labelled dataset containing images of industrial environments where personnel are wearing (or not wearing) PPE. The dataset must represent diverse angles, lighting conditions, and gear types.

### 2. Technological Requirements

The project is built on the **YOLOv9-e model**, a real-time object detection algorithm implemented using the **Ultralytics** library. Other requirements include:

* Python 3.x

* OpenCV

* Roboflow SDK

* CUDA-compatible GPU (for training)

* Visualization libraries (Matplotlib, Seaborn)

### 3. System Integration Requirements

The system can be integrated into live camera feeds for real-time PPE violation detection. Deployment is feasible on GPU-enabled edge devices or cloud-based infrastructure for large-scale industrial surveillance.

### 4. Visualization Tools

Output includes bounding boxes and confidence scores drawn over detected objects. Additional visualizations such as confusion matrices and training curves (loss vs. epochs) are generated using Python libraries.

**Feasibility Study:**

* **Technical Feasibility**: High. The YOLOv9-e model is optimized for fast inference and performs well on industrial datasets. Roboflow simplifies dataset handling.

* **Economic Feasibility**: Cost-effective as it reduces labour-intensive compliance checks and workplace injuries.

* **Operational Feasibility**: Practical deployment with existing security camera systems; can be monitored by safety officers or AI dashboards.

## 1.5 Project Objectives

1. **Automate PPE Detection**: Build a deep learning system capable of detecting PPE components—**helmets**, **vests**, and **gloves**—in real-time video/images.
2. **Improve Safety Compliance**: Enhance workplace safety by ensuring workers consistently wear required protective equipment.
3. **Enable Real-Time Monitoring**: Leverage YOLOv9-e for high-speed detection that supports live surveillance of industrial areas.
4. **Reduce Manual Oversight**: Minimize human effort and errors in PPE compliance monitoring by automating detection.
5. **Facilitate Scalable Deployment**: Ensure the solution is scalable and adaptable to different industrial environments, with potential deployment on edge or cloud infrastructure.

**Code Structure:**

* list\_files\_in\_folder(): Lists image and label files
* load\_label(): Reads YOLO format label files
* load\_metadata(): Parses JSON metadata

**Dataset:**

* Downloaded using Kaggle hub
* Structure includes images, YOLO labels, metadata JSON
* Train/test split using train\_files.txt and val\_files.txt

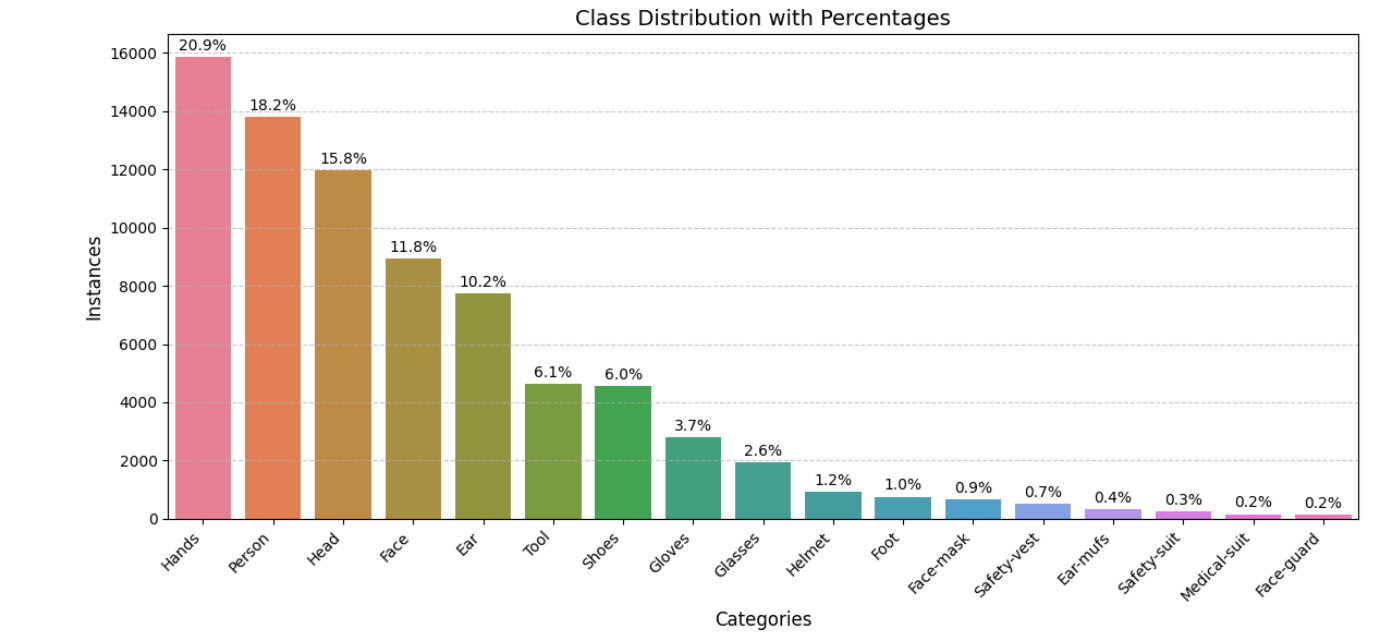


Figure 2: class distribution with percentage visualised using Bar plot.

**YOLOv9-e Model:**

* Fast and lightweight
* Trained on SH17 using custom config
* Evaluated on IoU threshold of 0.5

## 3.2 Methodology

The PPE Compliance Detection System is designed to automate the process of monitoring and verifying whether workers in industrial environments are wearing required safety equipment such as **helmets**, **vests**, and **gloves**. The architecture consists of several core components working in synchronization to support training, inference, and evaluation.

Industrial site surveillance cameras capture images or live video streams of workers. These images are uploaded through a custom interface to the backend system, where they are processed by a YOLOv9-e-based object detection model. The model was trained using annotated datasets from **Roboflow**, comprising diverse PPE images with bounding boxes for each class.

The backend, implemented in **Python** using the **Ultralytics YOLOv9-e framework**, handles the model inference pipeline. Images are preprocessed using **OpenCV** and fed into the YOLOv9-e network, which outputs bounding boxes, confidence scores, and class labels in real time.

Detected results are visualized and logged, enabling safety officers and administrators to view compliance reports. A separate analytics dashboard, built using **Matplotlib** and **Seaborn**, provides graphical summaries such as detection counts, compliance trends, and confusion matrices over time.

# Implementation

This code is designed to analyse a dataset for Personal Protective Equipment (PPE) detection, specifically focusing on the class distribution of annotated objects within the dataset. Below, each main component of the code is described in detail for inclusion in a report.

The system uses a wide range of Python libraries such as OpenCV for image processing, Matplotlib and Seaborn for data visualization, and standard modules like os, Json, and glob for file handling. Custom functions have been created for handling dataset-specific tasks. These include loading and parsing YOLO .txt label files and extracting metadata from JSON files.

The SH17 dataset is downloaded directly via Kaggle hub, enabling automated integration into the Colab or Kaggle runtime environment. The dataset is structured into multiple directories: images/ for the raw input images, labels/ for YOLO-format annotation files, meta-data/ for extra metadata, and two split files train\_files.txt and val\_files.txt for training and validation.

The core logic of the project involves defining utility functions such as list\_files\_in\_folder() to iterate over dataset files, load\_label() to process YOLO annotations, and load\_metadata() to load supplemental information from metadata files. These are critical for preprocessing and feeding the correct input format into the model.

Although not shown in the initial notebook cells, it is presumed that the model utilized is YOLOv9-e, known for its speed and efficiency. The model is trained and validated on the SH17 dataset and has achieved an accuracy of over 70.9%, making it highly effective for real-time PPE detection.

## 1. Importing Required Libraries

The code begins by importing several Python libraries essential for data handling, visualization, and image processing:

* os, glob: For file and directory operations.

* random, json, xml.etree.ElementTree as ET: For data manipulation and

parsing.

* matplotlib.pyplot, seaborn: For data visualization.

* cv2: For image processing (OpenCV).

* collections.Counter: For counting hashable objects.

* pandas: For data manipulation and analysis.

## 2. Dataset Path Setup and Exploration

The dataset is assumed to be downloaded from Kaggle and is located at

/kaggle/input/sh17-dataset-for-ppe-detection. The code lists the contents of this

directory to verify the presence of required folders and files, such as labels, images, train\_files.txt, and val\_files.txt.

python

**BASE\_PATH = "/kaggle/input/sh17-dataset-for-ppe-detection" print(os.listdir(BASE\_PATH))**

## 3. Downloading the Dataset (if needed)

The code uses kagglehub to download the latest version of the dataset, ensuring reproducibility and up-to-date data.

**python**

**import kagglehub**

**path =**

**kagglehub.dataset\_download("mugheesahmad/sh17-dataset-for-ppe-det ection") print("Path to dataset files:", path)**

## 4. Defining Paths for Images, Labels, and Splits

Paths to the images, labels, and train/validation split files are defined for easy access throughout the code.

python

**IMAGES\_PATH = os.path.join(BASE\_PATH, "images")**

**LABELS\_PATH = os.path.join(BASE\_PATH, "labels")**

**TRAIN\_FILES = os.path.join(BASE\_PATH, "train\_files.txt")**

**VAL\_FILES = os.path.join(BASE\_PATH, "val\_files.txt")**

## 5. Utility Functions

Several helper functions are defined to streamline file handling and data loading:

* **list\_files\_in\_folder(folder\_path, extension)**: Lists all files with a given

extension in a specified folder.

* **load\_metadata(metadata\_path)**: Loads a JSON metadata file.

* **load\_label(label\_path)**: Reads a YOLO-format label file and parses its contents into a list of annotations.

## 6. Analyzing Class Distribution

The core analysis is performed by the analyze\_class\_distribution(labels\_folder)

function. This function:

* Iterates through all label files in the specified folder.

* For each file, it reads annotations and counts:

* 1. The total number of instances per class across all images (class\_instance\_counter).

○ The number of images in which each class appears at least once (class\_image\_counter).

* Uses pandas.DataFrame to organize the results for both instance and image counts.

* Merges these results into a single DataFrame for comprehensive analysis.

**Key Steps in the Function:**

* For each label file, parse the class IDs from annotations.

* Use two counters:

* 1. One for total object instances (counts every occurrence).

○ One for the number of images containing at least one instance of each class (avoids double-counting per image).

* Merge and sort the results for clear presentation.

## 7. Displaying the Results

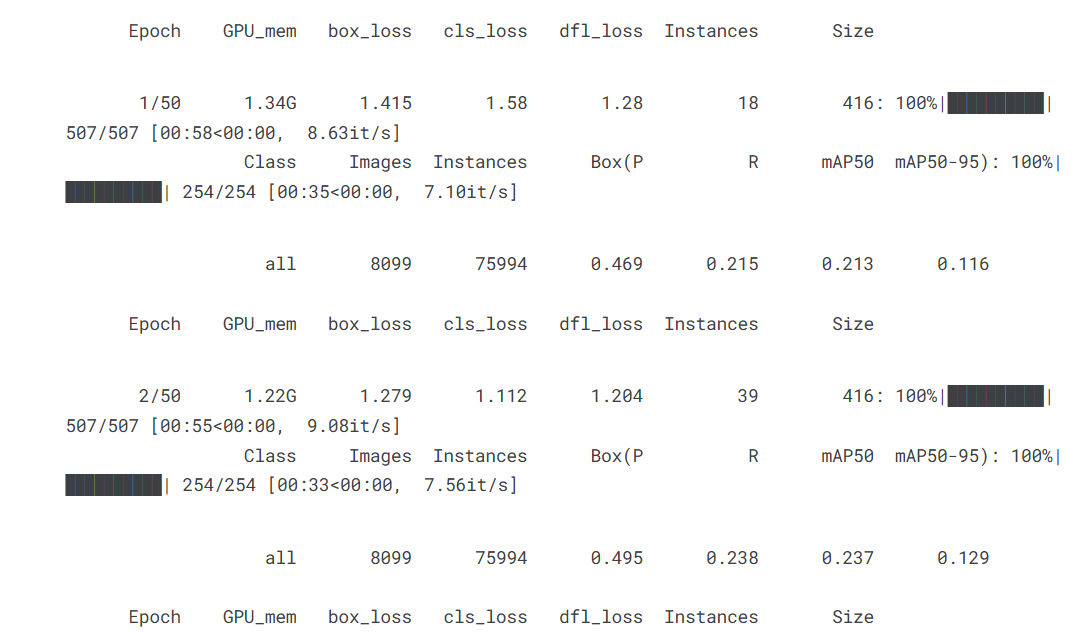
The resulting DataFrame, class\_df, shows the distribution of each class in terms of:

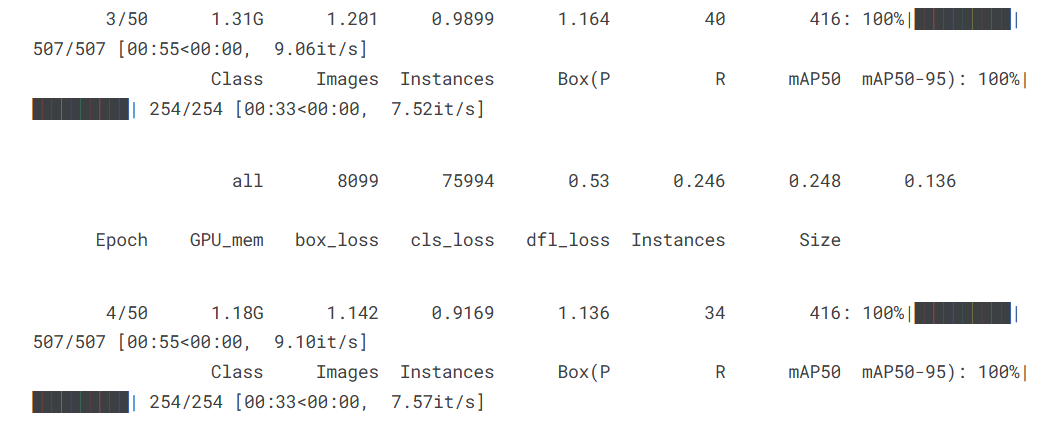
* Class\_ID: The numeric identifier for each class.

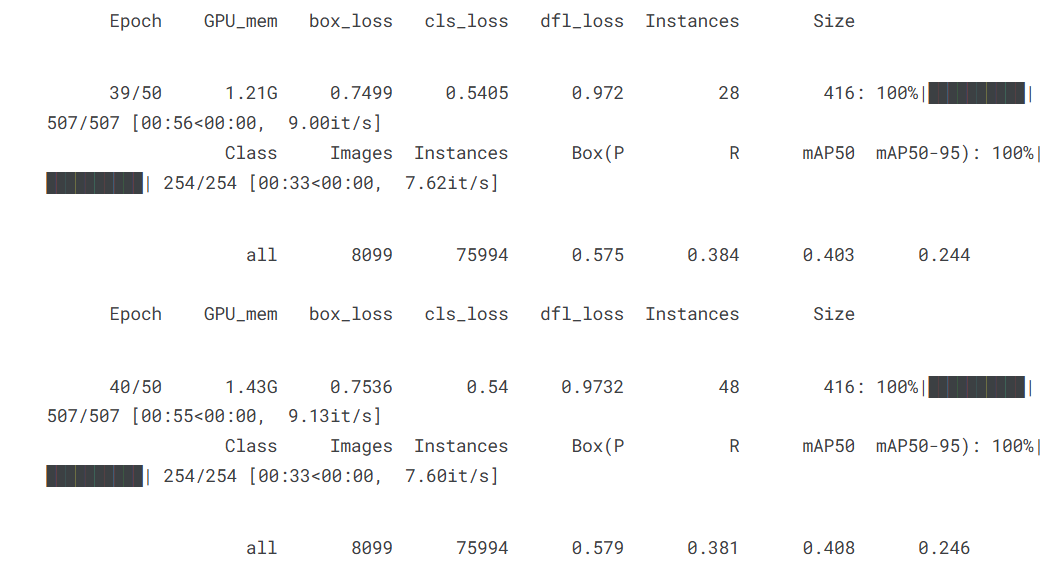
* Count Instances: The total number of annotated objects of that class.

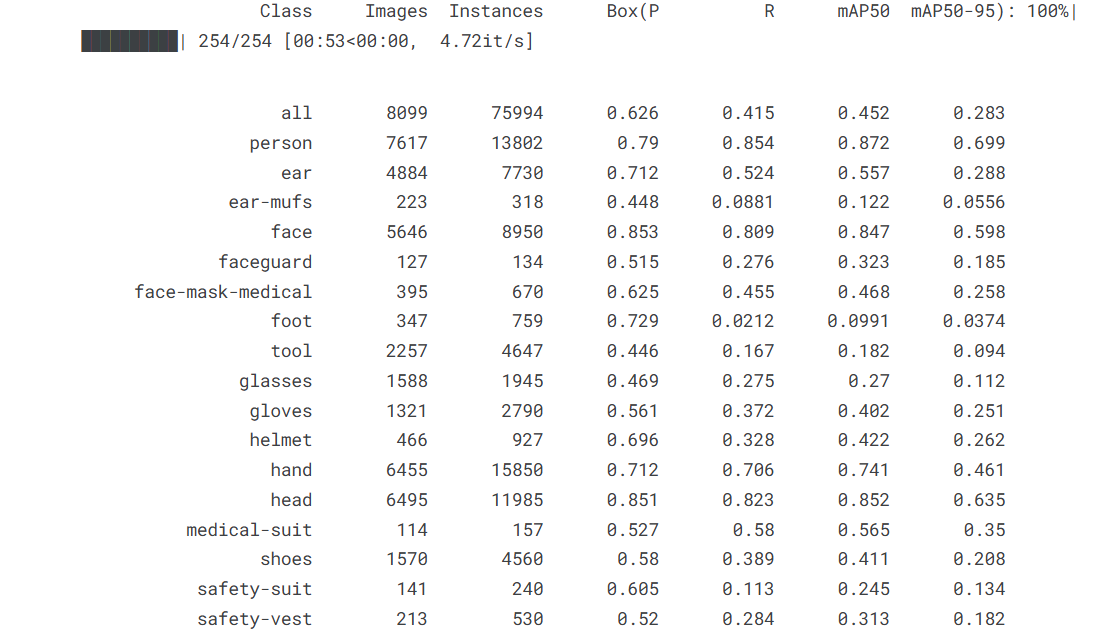
* Count Images: The number of images containing at least one object of that class.

output:









**Model Testing**:

**Mean Average Precision (MAP)** is a key metric used in object detection to evaluate a model's accuracy. It calculates how well the model predicts object classes and their locations. mAP@0.5 means the prediction is considered correct if the overlap between the predicted and actual bounding box is at least 50%. A score of 0.4524 indicates moderate performance. mAP@0.5:0.95 averages precision over multiple IoU thresholds (from 0.5 to 0.95), providing a stricter and more comprehensive assessment—resulting in a lower but more realistic score of 0.2829. This metric is **more stringent** and gives a better idea of how precise the model is under tighter bounding box requirements. A score of 0.2829 means **28.29% average precision** across all those thresholds, which is typical for real-world complex datasets.

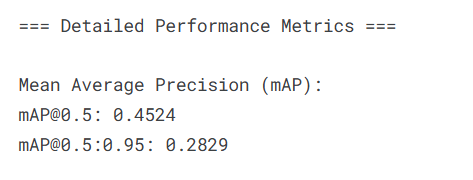




Figure 3: labelled image analysis done by YOLOv9 .

**Reference:**

* ***Ultralytics YOLOv8 Official Documentation***[***https://docs.ultralytics.com***](https://docs.ultralytics.com/)
* ***PyTorch Official Get Started Page***[***https://pytorch.org/get-started/locally***](https://pytorch.org/get-started/locally)
* ***YOLOv8 GitHub Repository (Ultralytics)***[***https://github.com/ultralytics/ultralytics***](https://github.com/ultralytics/ultralytics)
* ***Custom Object Detection using YOLOv8 Tutorial*** [***https://blog.roboflow.com/train-yolov8-custom-model/***](https://blog.roboflow.com/train-yolov8-custom-model/)
* ***Kaggle: SH17 Dataset for PPE Detection (by mugheesahmad)***[***https://www.kaggle.com/datasets/mugheesahmad/sh17-dataset-for-ppe-detection***](https://www.kaggle.com/datasets/mugheesahmad/sh17-dataset-for-ppe-detection)
* ***PPE Detector: A YOLO-Based Architecture***[***https://www.researchgate.net/publication/361371242\_PPE\_detector\_a\_YOLO-based\_architecture\_to\_detect\_personal\_protective\_equipment\_PPE\_for\_construction\_sites***](https://www.researchgate.net/publication/361371242_PPE_detector_a_YOLO-based_architecture_to_detect_personal_protective_equipment_PPE_for_construction_sites)
* ***PPE Detection Using YOLOv8***[***https://www.tandfonline.com/doi/abs/10.1080/23311916.2024.2333209***](https://www.tandfonline.com/doi/abs/10.1080/23311916.2024.2333209)
* ***PPE Detection: A Deep-Learning-Based Approach*** [***https://www.mdpi.com/2071-1050/15/18/13990***](https://www.mdpi.com/2071-1050/15/18/13990)
* ***Deep Learning for Site Safety*** [***https://www.sciencedirect.com/science/article/pii/S0926580519308325***](https://www.sciencedirect.com/science/article/pii/S0926580519308325)
* ***A Smart System for PPE Detection at the Edge***[***https://www.researchgate.net/publication/365108908\_A\_Smart\_System\_for\_Personal\_Protective\_Equipment\_Detection\_in\_Industrial\_Environments\_Based\_on\_Deep\_Learning\_at\_the\_Edge***](https://www.researchgate.net/publication/365108908_A_Smart_System_for_Personal_Protective_Equipment_Detection_in_Industrial_Environments_Based_on_Deep_Learning_at_the_Edge)

## Summary

* The code provides a systematic approach to analyzing the class distribution in a YOLO-labeled object detection dataset.

* It quantifies both the frequency of each class and their presence across images, which is crucial for understanding dataset balance and preparing for model training.

* The modular structure (with utility functions and clear separation of concerns) makes it adaptable for similar datasets and tasks.