## ****OBJECTIVE****

The objective of this project is to build a robust object detection system that can identify **Personal Protective Equipment (PPE)** such as **helmets**, **gloves**, and **vests** in real-time or uploaded videos. This is achieved using the **YOLOv11** (an advanced version of YOLOv8+) deep learning model, trained on a **custom Roboflow-annotated dataset**. The final system includes a **Gradio-based UI** for user interaction and supports both **video uploads** and **live webcam inference.**

**1. How YOLO Works**

YOLO (You Only Look Once) is a powerful, real-time object detection algorithm that treats object detection as a single regression task. YOLOv11 builds on previous YOLO versions, using a more optimized backbone neck and head design for enhanced feature extraction and multi-scale detection.

* Single-pass inference on full images – hence known as Only look once
* Grid-based prediction: YOLO divides the input image into an S×S times grid. Each grid cell is responsible for detecting objects whose centers fall within it.
* Detection of multiple classes with bounding boxes.

**Bounding Box**  
A bounding box is a rectangle that encloses an object in an image. YOLO predicts several bounding boxes per grid cell. Low-confidence boxes are discarded, and overlapping boxes are reduced using non-maximum suppression (NMS).

**Anchor Boxes**  
Anchor boxes are predefined box shapes that help YOLO detect objects of different sizes and shapes. The model predicts adjustments to these anchors to better fit the objects, improving accuracy for multiple objects in the same grid cell.

**2. YOLO Training Format**

To train YOLOv11, the dataset must be organized as follows:

**Directory Structure:**

/dataset

/images/train

/images/val

/images/test

/labels/train

/labels/val

/labels/test

data.yaml

**Label Format (per image):**

<class\_id> <x\_center> <y\_center> <width> <height>

* All coordinates are normalized (0 to 1).
* data.yaml contains paths and class names.

**3. Evaluation Metrics**

**IoU (Intersection over Union)**: IoU measures how much the predicted bounding box overlaps with the ground truth box. A prediction is considered **correct** (True Positive) if IoU ≥ a threshold (commonly 0.5 or 0.75). The higher the IoU threshold, the stricter the matching.

IOU = Area of overlap / Area of union

**Precision**: **Precision measures how many of the detected objects are** **actually correct**. High precision means fewer false alarms. In object detection, high precision indicates that most of the predicted bounding boxes match actual objects.

TP / (TP + FP)

**Recall**: Recall measures how many of the actual objects are successfully detected. High recall means most real objects are detected. In object detection, high recall means the model misses fewer objects.

TP / (TP + FN)

**Average Precision (AP):** Combines precision and recall across different thresholds or confidence levels. It’s the area under the Precision-Recall curve.

**Mean Average Precision (mAP):** Average of AP across multiple classes (or IoU thresholds). Common metric for object detection benchmarks.

 **TP (True Positive):**  
The number of correctly detected objects. That means the model predicted an object, and it really exists at that location.

 **FP (False Positive):**  
The number of incorrect detections. The model predicted an object, but there is no actual object (a false alarm).

 **FN (False Negative):**  
The number of missed detections. There was an object in the ground truth, but the model failed to detect it.

**4. Dataset Splits**

* **Training Set**: Model learns patterns (70–80%)
* **Validation Set**: Used during training for tuning (10–20%)
* **Test Set**: Used after training for final evaluation (10–20%)

Each split ensures accurate performance assessment and avoids overfitting.

**Overfitting:** occurs when a machine learning model learns not only the underlying patterns in the training data but also the noise and random fluctuations specific to that data. As a result, the model performs exceptionally well on the training set but poorly on new, unseen data.

* The model becomes **too complex**, capturing details that do not generalize beyond the training dataset.
* It **memorizes** training examples instead of learning generalizable features.
* Leads to **high accuracy on training data** but **low accuracy on validation or test data**.
* Overfitting reduces the model’s ability to make reliable predictions on real-world data.

**5. Libraries**

**1. Ultralytics :** The ultralytics library is the official implementation for the latest versions of the YOLO (You Only Look Once) object detection models, including YOLOv8 and YOLOv11. It provides high-level APIs for:

* + Loading pretrained or custom YOLO models
  + Training, validation, and inference
  + Visualizing detection outputs
  + Exporting models in different formats (ONNX, TorchScript, etc.)
  + The YOLO class from this library was used to load the trained model (best.pt) and perform inference on each video frame.

**2. OpenCV** : Open Source Computer Vision Library is a powerful toolkit for real-time computer vision. It is used to perform:

* + Video capture and playback
  + Frame-by-frame image processing
  + Drawing bounding boxes, displaying annotated frames
  + Keyboard input handling (e.g., exiting on pressing 'q')
  + cv2.VideoCapture() was used to load the input video, and cv2.imshow() was used to display detection results live. It also handles window management and termination control.

**3. NumPy :** Fundamental package for numerical computing in Python. It provides:

* + Efficient storage and manipulation of large arrays and matrices
  + Support for mathematical operations on array structures
  + Compatibility with OpenCV (as OpenCV images and frames are stored as NumPy arrays)
  + automatically leveraged for manipulating image matrices during preprocessing and postprocessing.

**6. Other Tools**

**1. PyTorch (torch)**

* The deep learning framework that underpins the entire Ultralytics YOLO engine. Automatically installed with ultralytics. It's essential for loading, training, and running models. It handles:
  + GPU acceleration (CUDA)
  + Training loops and backpropagation

## ****CONCLUSION****

The successful implementation of the **PPE Detection system using YOLOv11** involved understanding the YOLO architecture, structuring the dataset properly, training the model with appropriate evaluation metrics, and performing real-time inference. This setup enables practical applications in safety monitoring and industrial automation.