Ministry of Higher Education and Scientific Research


National Engineering School of Computer Science, Sidi-Bel-Abbès

# Personal Protective Equipment (PPE) Detection

Using Computer vision & Deep Learning Models

**Prepared by:**

OUARETH Mohamed El Salih
BADEREDDINE Haithem
DADOUNE Ikram
RAMLA Yassine

**Supervised by:**
KHALDI Belkacem

**Academic Year 2024-2025**

**Abstract**

Real-time object detection has become essential for automating safety monitoring tasks in industrial environments. The **AI Safety** project addresses this critical need by developing an intelligent system for Personal Protective Equipment (PPE) compliance monitoring. A critical application is ensuring worker safety through automated monitoring of Personal Protective Equipment (PPE) compliance in hazardous work areas. Traditional monitoring systems rely on continuous video stream analysis from surveillance cameras to assess PPE usage in real time, triggering automatic acoustic or visual alarms when workers are detected without appropriate protective equipment.

The **AI Safety** project presents a comprehensive evaluation of four state-of-the-art deep learning models for real-time PPE detection: **YOLOv8x,YOLOv8-Large, YOLOv8-Medium, SSD640, and Faster R-CNN ResNet50 FPN v2.**

We developed our system with the four models were fine-tuned to detect critical PPE items including person,safety helmets, high-visibility vests, and protective gloves. Our experimental evaluation encompasses two key phases: **first**, we conducted comprehensive performance comparisons of all four models in terms of detection accuracy, precision, recall, and inference latency under controlled conditions. **Second**, we deployed best model on the embedded system to evaluate real-world performance metrics, including system throughput measured by the number of video frames processed per second, power consumption, and overall system responsiveness.

The results demonstrate the feasibility of deploying sophisticated deep learning models for PPE detection at the edge, providing reliable safety monitoring without dependence on cloud infrastructure. Our comparative analysis offers insights into the trade-offs between model complexity, detection performance, and computational requirements, enabling informed selection of appropriate models based on specific deployment constraints and safety requirements.

# Contents

# 1 Introduction

## 1.1 Background and Motivation

Personal Protective Equipment (PPE) is critical in ensuring workplace safety, particularly in high-risk environments such as construction sites, factories, and industrial facilities. PPE, including helmets, vests, and gloves, protects workers from hazards, while access control measures ensure that only authorized personnel enter restricted areas. Manual monitoring of PPE compliance and access control is labor-intensive and prone to errors, necessitating automated solutions. Advances in computer vision and deep learning have enabled the development of robust systems for real-time PPE detection and person identification, improving safety and efficiency.

This project addresses the need for an automated PPE detection system capable of identifying helmets, vests, gloves, and distinguishing between authorized and unauthorized persons. By leveraging state-of-the-art deep learning models, we aim to enhance workplace safety through accurate and efficient monitoring.

Figure 1: Personal protective equipment (PPE) types [2].

## 1.2 Objectives

The primary objectives of this project are:

- To develop and evaluate multiple deep learning models for PPE detection, including YOLOv8x,YOLOv8m, YOLOv8l, SSD640, Faster R-CNN-ResNet50-FPN-v2 and a DeepFace

- To compare the performance of these models in terms of accuracy, speed, and robustness in detecting helmets, vests, gloves, and identifying authorized/unauthorized persons.
- To implement the best-performing model on a React-based platform for real-time PPE monitoring and user interaction.
- To contribute to workplace safety by providing a reliable, automated solution for PPE compliance and access control.

## 1.3 Scope and Methodology

The project focuses on detecting PPE components (helmets, vests, gloves) and classifying individuals as authorized or unauthorized using computer vision techniques. We evaluated four deep learning models:

- **YOLOv8x,YOLOv8m and YOLOv8l**: Advanced object detection models known for their balance of speed and accuracy.
- **Faster R-CNN-ResNet50-FPN-v2**: A custom convolutional neural network leveraging transfer learning for feature extraction and classification.
- **SSD640**: A single-shot detector optimized for real-time applications.
- **DeepFace**: a deep learning framework for face recognition models to verify identities by comparing detected faces against a database of authorized personnel.

These models were trained and tested on a dataset containing images of workers with and without PPE, as well as authorized and unauthorized individuals. The best-performing model was integrated into a React-based web application to provide a user-friendly interface for real-time monitoring.

# 2 Data Overview

## 2.1 Data Sources

The dataset for this project was compiled from four sources obtained through Roboflow, a platform providing annotated datasets for computer vision tasks. Three datasets were dedicated to specific Personal Protective Equipment (PPE) classes:

- **Helmet Dataset**: Contains images of workers with and without helmets, annotated to identify helmet presence.
- **Vest Dataset**: Includes images of workers with and without safety vests, annotated for vest detection.
- **Glove Dataset**: Comprises images of workers with and without gloves, annotated for glove detection.

The fourth dataset was a custom collection of 1,300 images, gathered and manually annotated by our team to enhance the representation of PPE components in diverse workplace scenarios.

## 2.2 Data Augmentation and Annotation

Upon reviewing the Roboflow datasets, we observed a limited number of images labeled for all PPE classes, including "no helmet," "no vest," and "no gloves," which could hinder the models' ability to accurately detect the absence of these components. To address this, we performed data augmentation and annotation:

- **Custom Dataset**: The 1,300-image custom dataset was manually annotated to include labels for helmets, vests, and gloves, ensuring balanced representation of both presence and absence of these PPE components.
- **Cross-Annotation**: We revisited the three Roboflow datasets and added cross-annotations to include labels for the absence of other PPE components (e.g., "no vest" ,"no helmet" and "no gloves") to improve dataset diversity.
- **Augmentation Techniques**: The combined dataset, initially containing 5,181 images, was augmented to 13,000 images using techniques in Roboflow, including 30% rotation, 10% blur, and the addition of noise to enhance model robustness.

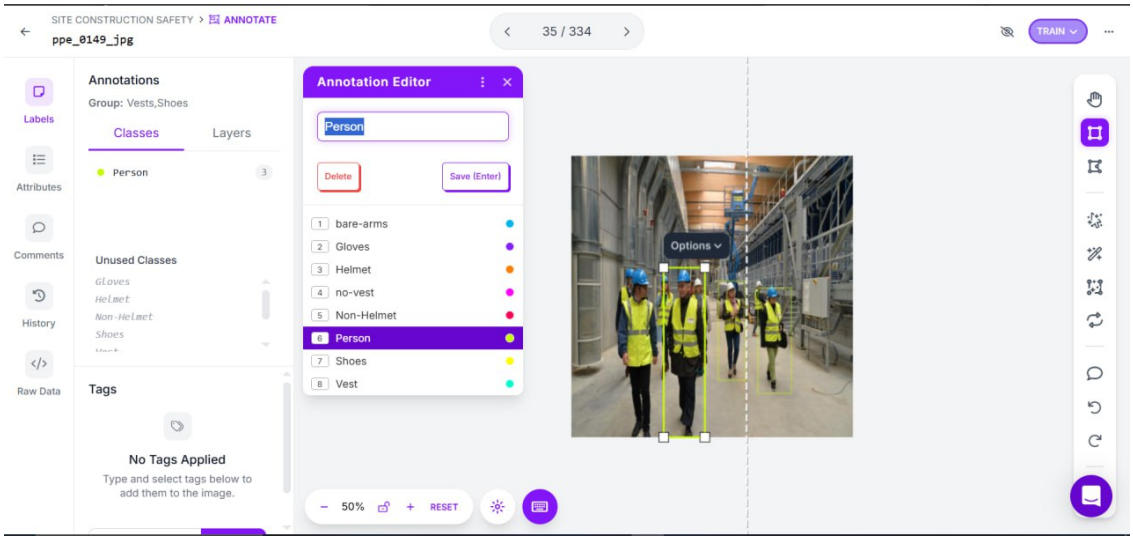



Figure 2: Screenshot from Roboflow illustrating the annotation process for PPE components.

## 2.3 Data Utilization

The final dataset of 11271 images was split into training, validation, and test sets to evaluate the four deep learning models (YOLOv8m, YOLOv8l, SSD640, and ResNet50-based CNN). The split consisted of 9159 images for training, 1578 images for validation, and 534 for testing. All data management, including annotation and augmentation, was performed using Roboflow to ensure consistency and reproducibility.

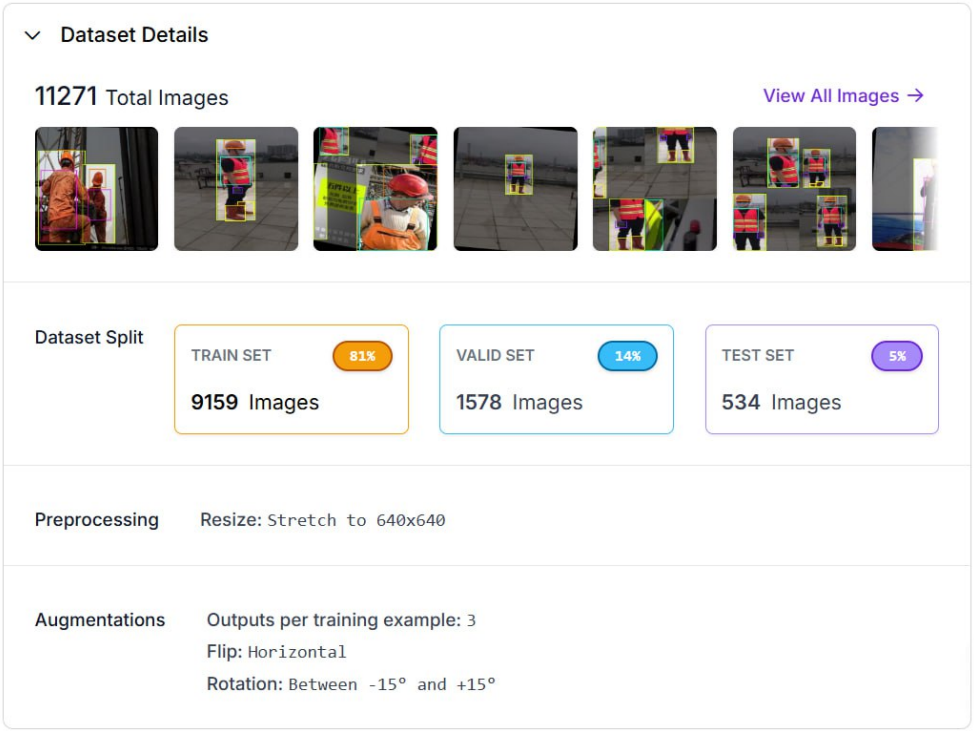



Figure 3: Screenshot from Roboflow showing dataset details, including total images and split into training, validation, and test sets.

| COLOR | CLASS NAME | COUNT |
|---|---|---|
| | Gloves | 10,325 |
| | Helmet | 8,425 |
| | Non-Helmet | 1,937 |
| | Person | 14,963 |
| | Shoes | 9,082 |
| | Vest | 11,702 |
| | bare-arms | 3,746 |
| | no-vest | 2,632 |

Figure 4: Classes Distribution

# 3 Methodology

To achieve the objective mentioned in Introduction the methodology adopted in this study is as shown in figure 5. The methodology includes data collection, preparation, model training, and performance evaluation

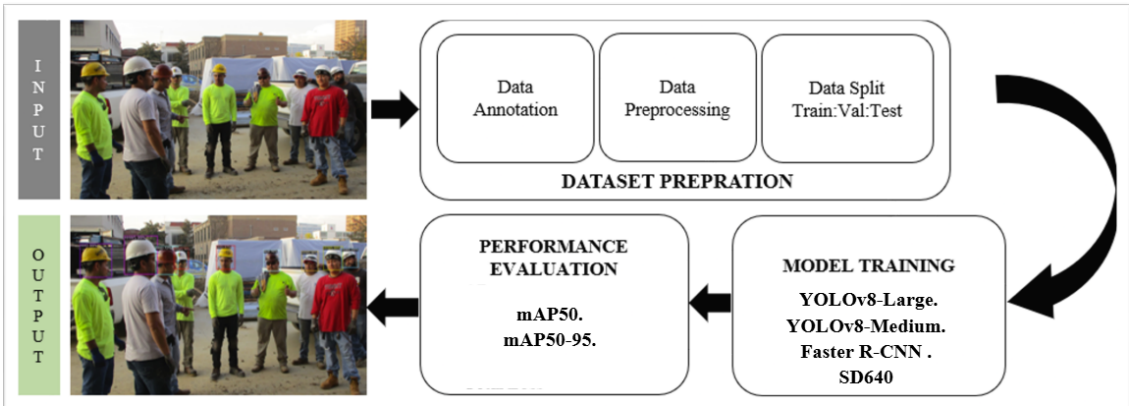



Figure 5: Object Detection Methodology

# 4 Models Architectures

## 4.1 YOLOv8

### 4.1.1 Overview

YOLOv8, developed by Ultralytics, is a state-of-the-art object detection model pushing the boundaries of speed, accuracy, and user-friendliness. The acronym YOLO, which stands for "You Only Look Once," enhances the algorithm's efficiency and real-time processing capabilities by simultaneously predicting all

bounding boxes in a single network pass. In comparison, several other object detection techniques require the detection process to go through several phases or processes. The popular YOLOv5 architecture is expanded upon in YOLOv8, which offers enhancements in these areas. The primary distinction between the YOLOv8 model and its predecessor is the utilization of anchor-free detection, which expedites the Non-Maximum Suppression (NMS) post-processing. YOLOv8 can identify and locate objects in images and videos with impressive speed and precision and tackles tasks like image classification and instance segmentation.

### 4.1.2 YOLOv8 Variants

| Model Variant | d(depth_multiple) | w(width_multiple) | mc(max_channels) |
|---|---|---|---|
| n | 0.33 | 0.25 | 1024 |
| s | 0.33 | 0.50 | 1024 |
| m | 0.67 | 0.75 | 768 |
| l | 1.00 | 1.00 | 512 |
| xl | 1.00 | 1.25 | 512 |

Table 1: YOLOv8 Model Variants and Their Specifications [3]

All of these models belong to the YOLOv8 family, each variant offers different trade-offs between accuracy, speed, and model size. The variants are divided based on the difference in the value of the parameters like **depth_multiple** (d), **width_multiple** (w), and **max_channel** (mc).

- **depth_multiple ($d$):** This parameter determines how many Bottleneck Blocks are used in the C2f block. This scales the number of layers in the network. A value less than 1 reduces the depth (fewer layers), making the model smaller and faster but potentially less accurate. Conversely, a value greater than 1 increases the depth (more layers), leading to a larger and potentially more accurate model but slower to run.
- **width_multiple ($w$):** This scales the number of channels in the convolutional layers. A value less than 1 thins the network (fewer channels), resulting in a smaller and faster model but potentially sacrificing some accuracy. On the other hand, a value greater than 1 widens the network (more channels), creating a larger and potentially more accurate model but requiring more processing power.
- **max_channels ($mc$):** This parameter sets an upper limit on the number of channels allowed in the network. It is a safety measure to prevent the model from becoming too wide (too many channels) especially when **width_multiple** is set high. This can help control the model size and prevent overfitting.

The YOLOv8 model types include:

- **n:** Nano model; smallest size, fastest inference, lowest accuracy.
- **s:** Small model; good balance between speed and accuracy.
- **m:** Medium model; higher accuracy with moderate inference speed.
- **l:** Large model; highest accuracy, slower inference.
- **xl:** Extra-large model; best accuracy for resource-intensive applications.

Blocks Used in YOLOv8 Architecture

Before taking a deep dive into the architecture of YOLOv8, we must understand the basic blocks used in the architecture.

### 4.1.3 Convolutional Block (Conv Block)

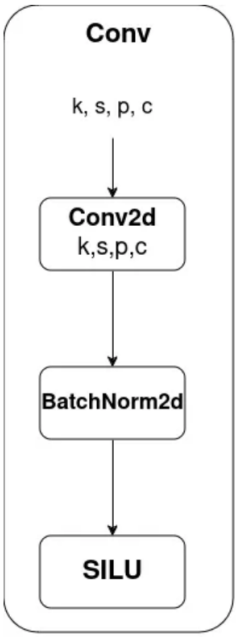



Figure 6: Convolutional Block (Conv Block) used in YOLOv8[3]

It is the most basic block in the architecture, which consists of the `Conv2d` layer, `BatchNorm2d` layer, and the SiLU activation function.

**Conv2d Layer:** Convolution is a mathematical operation that involves sliding a small matrix (called a kernel or filter) over the input data, performing element-wise multiplication, and summing the results to produce a feature map. The "2D" in Conv2D refers to the fact that the convolution is applied in two spatial dimensions, typically height and width.

- **k:** Number of filters or kernels. It represents the depth of the output volume, and each filter is responsible for detecting different features in the input.

- **s:** Stride. It is the step size at which the filter/kernel slides over the input. A larger stride reduces the spatial dimensions of the output volume.
- **p:** Padding. Padding is the additional border of zeros added to the input on each side. It helps preserve spatial information and can be used to control the spatial dimensions of the output volume.
- **c:** Number of channels in the input. For example, in an RGB image, $c = 3$ (one channel for each color).

**BatchNorm2d Layer:** Batch Normalization (BatchNorm2d) is a technique used in deep neural networks to improve training stability and convergence speed. In the context of convolutional neural networks (CNNs), the BatchNorm2d layer specifically applies batch normalization to 2D inputs, which are typically the outputs of convolutional layers. It ensures that the numbers going through the network aren't too big or too small. This helps in preventing problems during training.

**SiLU Activation Function:** SiLU, which stands for Sigmoid Linear Unit, is an activation function used in neural networks. It is also known as the Swish activation function. The SiLU activation function is defined as follows:

$$\text{SiLU}(x) = x \cdot \sigma(x), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The key characteristic of SiLU is that it allows for smooth gradients, which can be beneficial during the training of neural networks. Smooth gradients can help avoid issues like vanishing gradients, which can impede the learning process in deep neural networks.

#### 4.1.4 Bottleneck Block

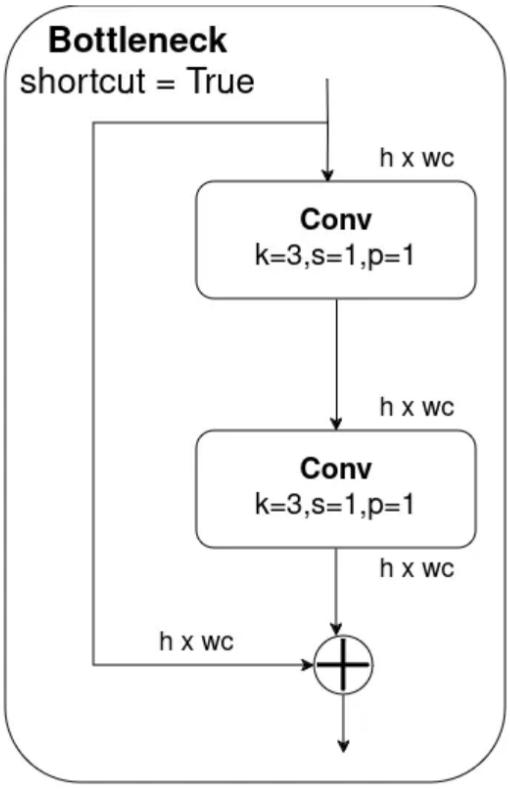



Figure 7: Bottleneck Block Architecture[3]

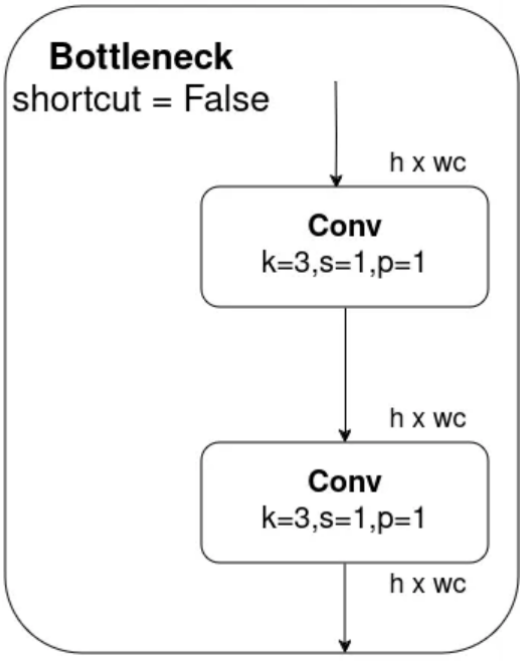



Figure 8: Bottleneck Block with Shortcut Connection[3]

The bottleneck block consists of the Conv Block with a shortcut connection. If the shortcut=true then the shortcut is implemented in the bottleneck block else the input is passed through two Conv Blocks in a series.

**Shortcut Connection:** The shortcut connection, also known as a skip connection or residual connection, is a direct connection that bypasses one or more layers in the network. It allows the gradient to flow more easily through the network during training, addressing the vanishing gradient problem and making it easier for the model to learn.

In the specific context of a bottleneck block, the shortcut connection allows the model to bypass the convolutional blocks if necessary. This way, the model can choose to use the identity mapping provided by the shortcut, making it easier to learn the identity function when needed. The inclusion of a shortcut connection enhances the ability of the model to learn complex representations and improves the training of deep CNNs preventing vanishing gradient problems.

**What is the vanishing gradient problem?** The vanishing gradient problem is a challenge that arises during the training of deep neural networks, particularly in architectures with many layers. It occurs when the gradients of the loss function concerning the parameters (weights) of the network become extremely small as they are backpropagated from the output layer to the input layer during the training process.

### 4.1.5 C2f Block

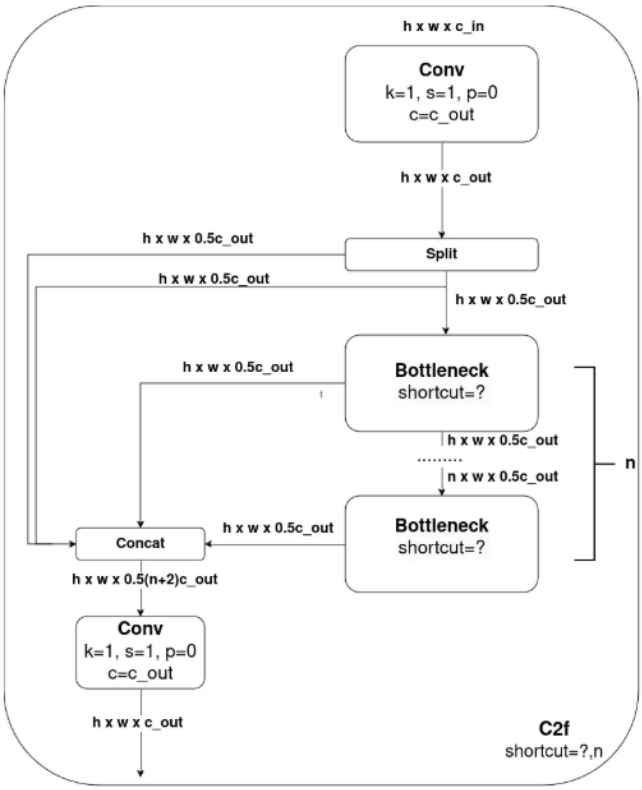



Figure 9: C2f Block Architecture[3]

The C2f block consists of a convolutional block which then the resulting feature map will be split. One feature map goes to the Bottleneck block whereas the other goes directly to the Concat block. In the C2f block, the number of the Bottleneck blocks used is defined by the `depth_multiple` parameter of the model. At the end, the feature map from the bottleneck block and the split feature map are concatenated and inputted into a final convolutional block.

#### 4.1.6 SPPF Block (Spatial Pyramid Pooling Fast)

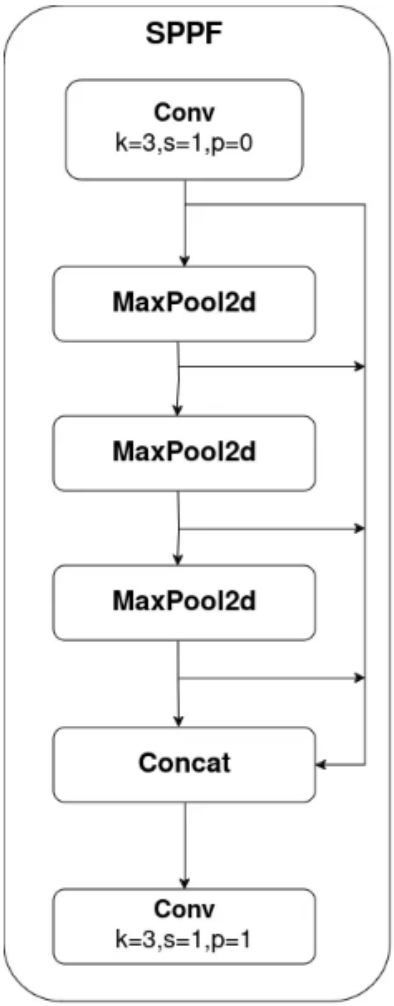



Figure 10: SPPF Block Architecture[3]

The SPPF Block consists of a convolutional block followed by three MaxPool2d layers. Every resulting feature map from the MaxPool2d layer is then concatenated at the end and fed to a convolutional block.

The basic idea behind Spatial Pyramid Pooling is to divide the input image into a grid and pool features from each grid cell independently, allowing the network to handle images of different sizes effectively.

In essence, Spatial Pyramid Pooling enables neural networks to work with images of different resolutions by capturing multi-scale information through pooling operations at different levels of granularity.

This can be particularly useful in tasks such as object recognition, where objects may appear at different scales within an image.

While SPP offers advantages, it can be computationally expensive. SPP-Fast addresses this by using a simpler pooling scheme. Instead of using multiple pooling levels with different kernel sizes, SPP-Fast might use a single fixed-size kernel for pooling, reducing the number of computations needed. SPP-Fast offers a trade-off between accuracy and speed.

**MaxPool2d Layer:** Pooling layers are used to downsample the spatial dimensions of the input volume, reducing the computational complexity of the network and extracting dominant features. Max pooling is a specific type of pooling operation where, for each region in the input tensor, only the maximum value is retained, and the other values are discarded.

In the case of MaxPool2d, the pooling is applied in both the height and width dimensions of the input tensor. The layer is defined by specifying parameters such as the size of the pooling kernel and the stride. The kernel size determines the spatial extent of each pooling region, and the stride determines the step size between successive pooling regions.

### 4.1.7 Detect Block

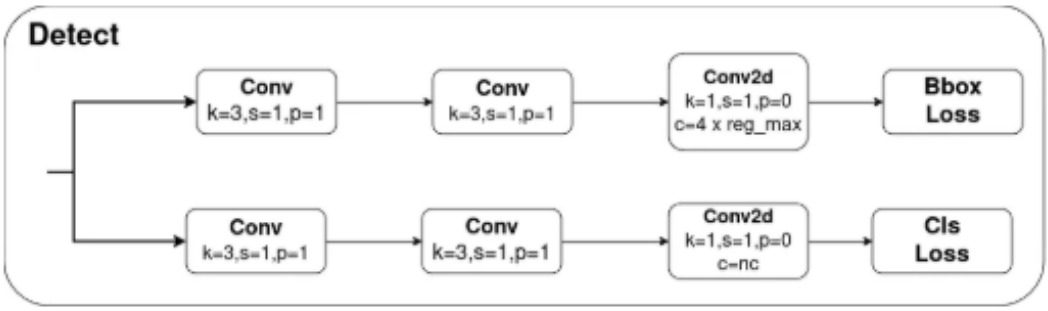



Figure 11: Detect Block Architecture[3]

The Detect Block is responsible for the detection of the objects. Unlike in previous versions of YOLO, YOLOv8 is an anchor-free model which means it predicts directly the center of an object instead of the offset from a known anchor box. Anchor-free detection reduces the number of box predictions, which speeds up complicated post-processing steps that sift through candidate detections after inference.

The Detect Block contains two tracks. The first track is for bounding box predictions and the second track is for class predictions. Both tracks contain two convolutional blocks followed by a single Conv2d layer which gives the Bounding Box loss and Class Loss respectively.

### 4.1.8 YOLOv8 Architecture Explained

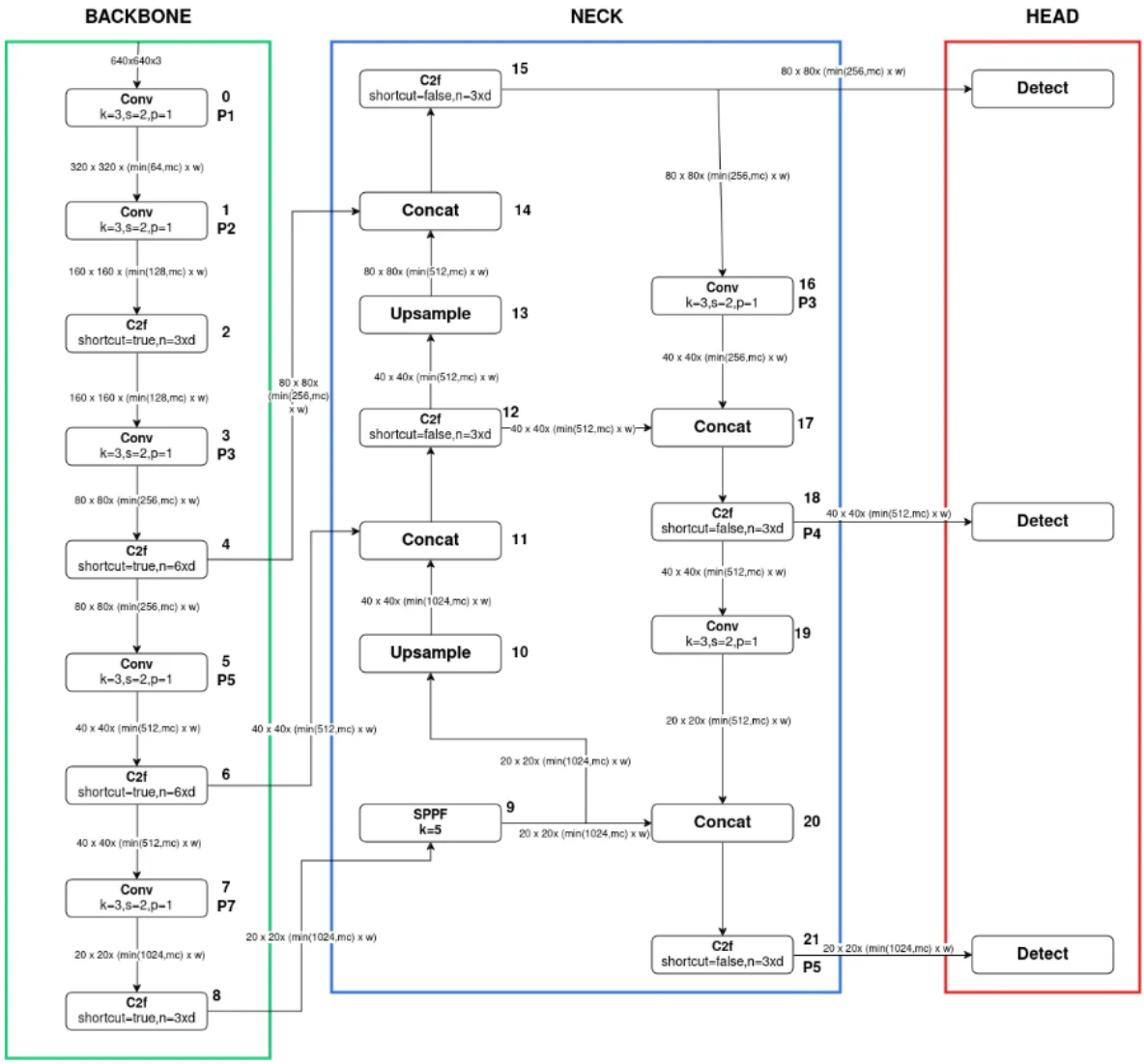



Figure 12: Complete YOLOv8 Architecture Overview[3]

The YOLOv8 architecture consists of three main sections: **Backbone**, **Neck**, and **Head**.

- **Backbone:** The deep learning architecture that acts as a feature extractor for the input image. It learns hierarchical representations of visual features from low-level edges to high-level object parts.
- **Neck:** This module combines features extracted from various levels of the Backbone. It typically enhances and aggregates these features to improve the detection performance across different object scales.
- **Head:** Responsible for producing the final outputs of the model, which include object class predictions and the corresponding bounding box coordinates.

### 4.1.9 Backbone Section

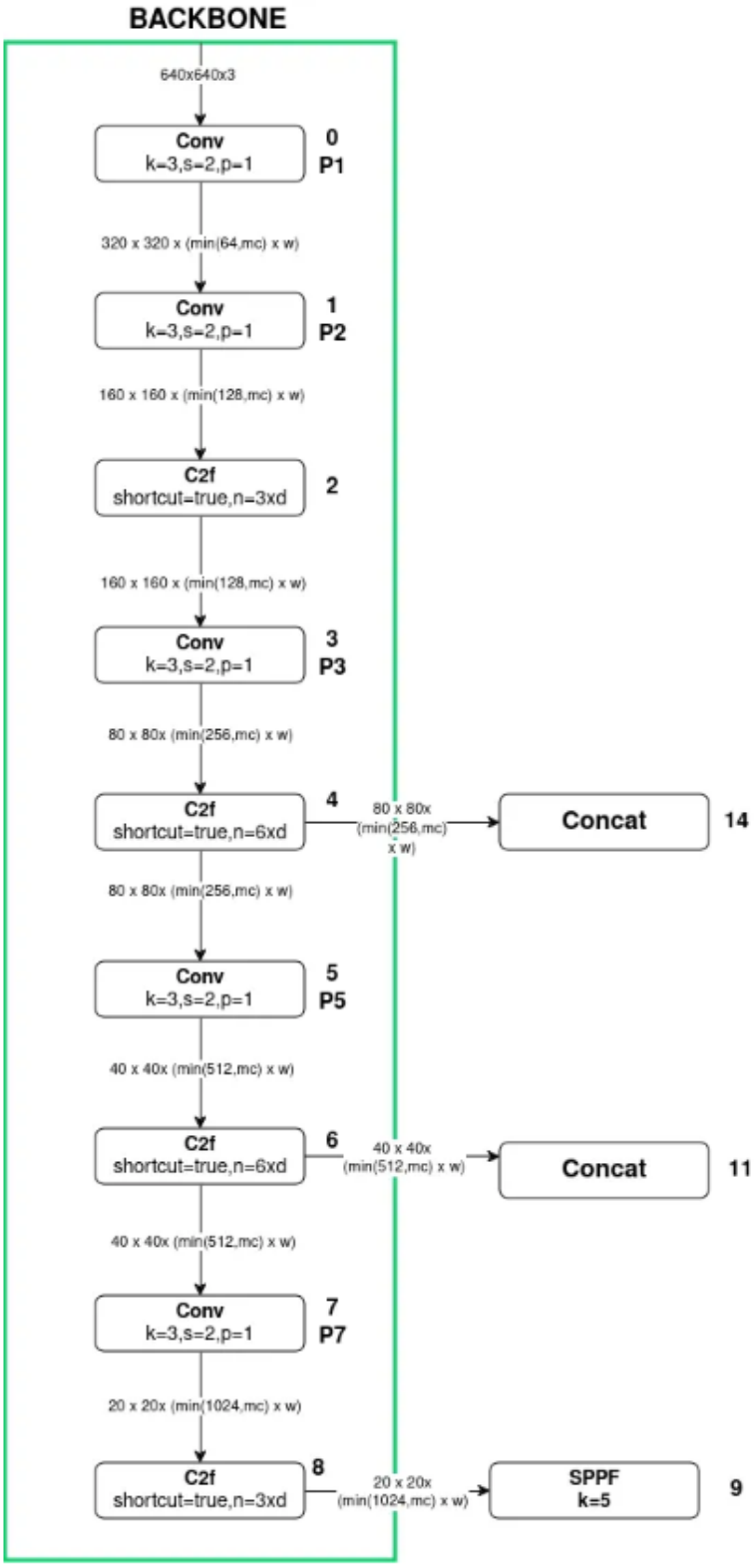



Figure 13: YOLOv8 Backbone Architecture[3]

In **Block0**, the processing starts with the input image size of **640 × 640 × 3** which is fed to the convolutional block with kernel **size 3**, **stride 2**, and **padding 1**. The spatial resolution is reduced when **stride = 2** is used. The convolutional block produces the feature map of **320 × 320** because the kernel moves in **2-pixel** increments.

To obtain the output channel of the convolution block, the following formula is used:

$$\text{Output Channels} = \min(64, mc) \times w \tag{2}$$

Here:

- **64** is the base output channel.
- **mc** is the max channel.
- **w** is the width multiple.

For example, if we are using the "n" variant YOLOv8 model then our **final output channel becomes** $= \min(64, 1024) \times 0.25 = 64 \times 0.25 = 16$

Likewise, this operation is calculated in every convolutional block present in the architecture.

**Block 2** is a C2f block that contains two parameters i.e., **shortcut** and n. Here, the shortcut is the boolean parameter that denotes if the Bottleneck block utilizes the shortcut or not. If the value of the shortcut = true then the bottleneck block inside the C2f block utilizes the shortcut else it doesn't.

Here, n determines how many bottleneck blocks are used inside the C2f block. In the case of **Block 2**, **n** is given by:

$$n = 3 \times d \tag{3}$$

where **d = depth multiple**

For example, if we are using the "n" variant YOLOv8 model the **depth_multiple** of the "n" type YOLOv8 model is 0.33 so, the number of bottleneck blocks used inside the C2f becomes $\mathbf{n} = 3 \times 0.33 = 0.99$, i.e., 1 bottleneck block is used.

In the C2f block the resolution of the feature map and the output channel is unchanged.

In **Block 9**, the SPPF Block is used after the last convolution layer of the C2f block in the Backbone.

The main function of the SPPF block is to generate the fixed feature representation of the object in various sizes in an image without resizing the image or introducing spatial information loss.

### 4.1.10 Neck and Head Section

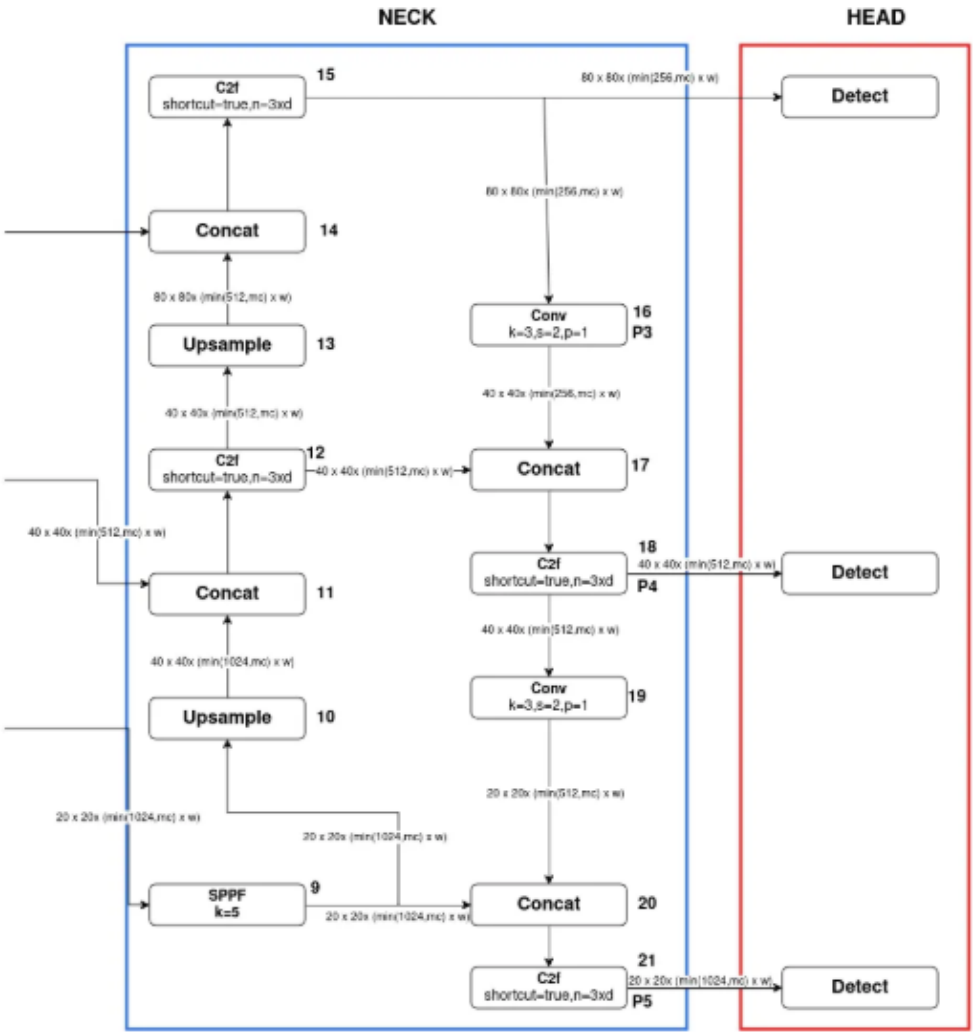



Figure 14: YOLOv8 Neck and Head Architecture[3]

**The Neck Section** The neck section is responsible for upsampling the feature map and combining the features acquired from the various layers of the Backbone section.

- The upsample layer present in the Neck section simply increases the feature map by double without making any changes in the output channel.
- Concat Block sums off the output channels of the blocks that are being concatenated without any change in resolution.

**The Head Section** The head section is responsible for predicting the classes and the bounding box of the objects which is the final output produced by the object detection model.

- The first Detect block in the Head section specializes in detecting small objects that are inputted from the C2f block present in Block 15.

- The second Detect block in the Head section specializes in detecting medium-sized objects which is inputted from the C2f block present in Block 18.
- The third Detect block in the Head section specializes in detecting large objects that are inputted from the C2f block present in Block 21.

## 4.2 Faster R-CNN-ResNet50-FPN-v2

### 4.2.1 Overview

Faster R-CNN is a two-stage object detection architecture that combines region proposal generation with object classification and localization. This implementation uses ResNet50 as the backbone with Feature Pyramid Network (FPN) v2 for enhanced multi-scale feature extraction.

### 4.2.2 Architecture Components

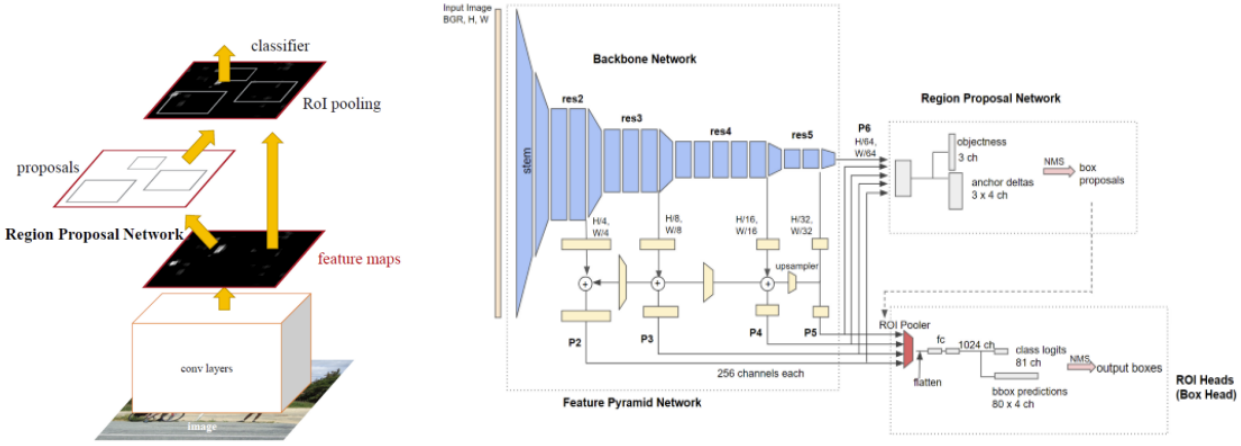



Figure 15: Faster R-CNN-ResNet50-FPN-v2 Architecture[5].

**Backbone Network: ResNet50** The ResNet50 backbone serves as the feature extraction foundation of the architecture.

**Structure:**

- **Input Layer**: Accepts RGB images of variable sizes (typically resized to 640×640 pixels)
- **Convolutional Layers**: 50 layers organized into residual blocks
- **Residual Connections**: Skip connections that enable gradient flow and prevent vanishing gradient problem
- **Bottleneck Design**: Uses 1×1, 3×3, 1×1 convolution pattern to reduce computational complexity

**Key Features:**

- **Depth**: 50 convolutional layers providing rich feature representation
- **Residual Learning**: Identity mappings allow training of very deep networks

- **Batch Normalization**: Applied after each convolution for stable training
- **Output Feature Maps**: Generates multi-scale feature maps at different resolutions (C2, C3, C4, C5)

**Feature Pyramid Network (FPN) v2** The FPN v2 enhances the original FPN design for better multi-scale object detection.

**Architecture Details:**

- **Bottom-up Pathway**: Utilizes ResNet50's hierarchical feature maps (C2, C3, C4, C5)
- **Top-down Pathway**: Upsamples higher-level features and combines with lower-level features
- **Lateral Connections**: 1×1 convolutions align channel dimensions between pathways
- **Feature Fusion**: Element-wise addition of upsampled and lateral features

**FPN v2 Improvements:**

- **Enhanced Feature Alignment**: Better spatial alignment of features across scales
- **Improved Gradient Flow**: Optimized connections for better backpropagation
- **Balanced Feature Learning**: More uniform feature learning across pyramid levels

### 4.2.3 Two-Stage Detection Process

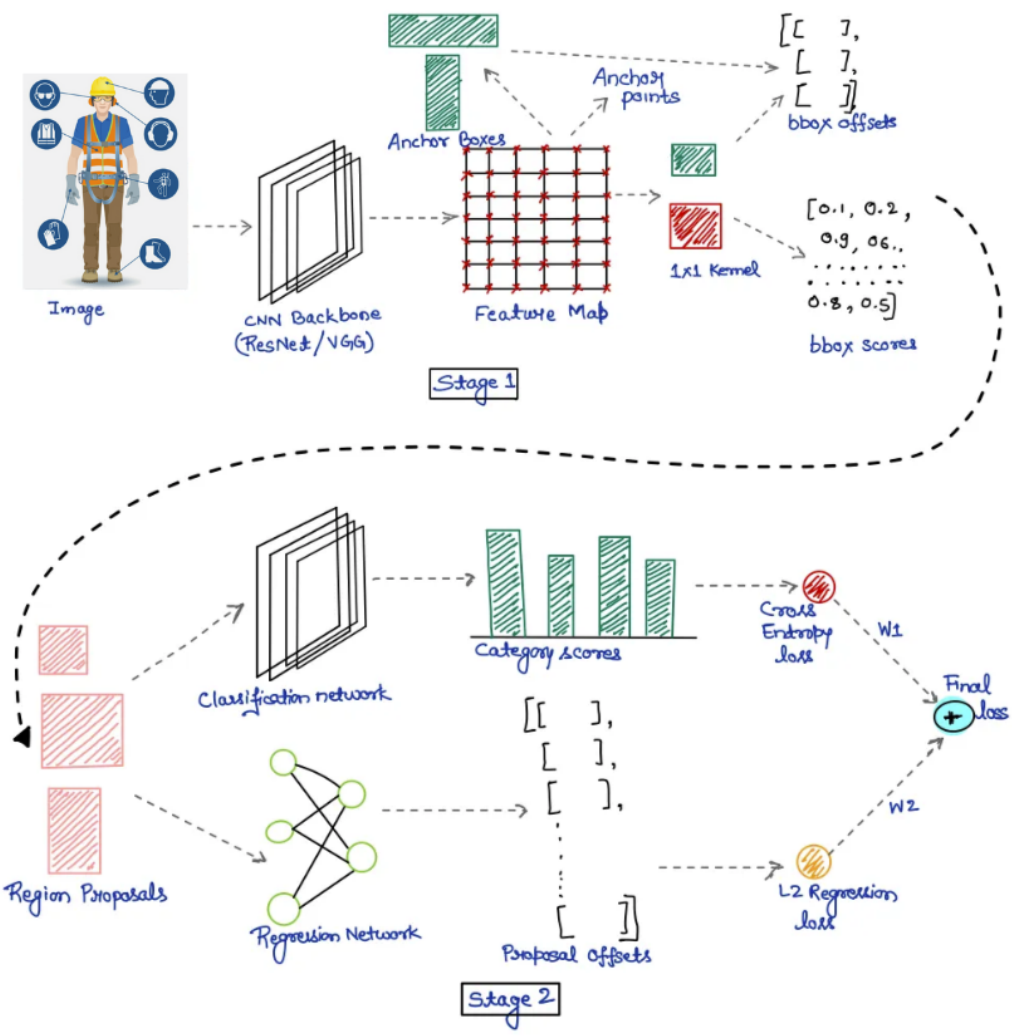



Figure 16: Two-Stage Detection Process

**Stage 1: Region Proposal Network (RPN)** The RPN proposes potential regions in the image that might contain objects.

**Components:**

*Anchor Generation*

- **Anchor Boxes**: Boxes of different sizes and shapes placed over points on the feature map
- **Multiple Scales**: Creates anchors at multiple scales (128, 256, 512 pixels)
- **Aspect Ratios**: Different aspect ratios (0.5, 1.0, 2.0) for various object shapes
- **Multi-level**: Operates on each level of the FPN pyramid

*Classification Head*

- **Binary Classification**: Predicts whether each anchor box contains an object (foreground) or background
- **Positive Anchors**: Boxes with high Intersection over Union (IoU) with ground truth objects
- **Negative Anchors**: Boxes with little or no overlap with objects

*Regression Head*

- **Bounding Box Refinement**: Predicts offsets to adjust anchor boxes for better alignment with actual objects
- **Coordinate Adjustments**: Refines position, width, and height of anchor boxes

**Loss Functions:**

- **Classification Loss**: Binary cross-entropy loss to distinguish foreground from background
- **Regression Loss**: Smooth L1 loss for bounding box coordinate refinement

**Post-processing:**

- **Non-Maximum Suppression (NMS)**: Removes redundant proposals with IoU threshold of 0.7
- **Proposal Filtering**: Selects top-k proposals for the next stage

**Stage 2: Object Classification and Box Refinement** The second stage uses proposed regions to predict object classes and refine bounding boxes.

*RoI (Region of Interest) Processing*

- **RoI Align**: Uses bilinear interpolation for sub-pixel accuracy in feature extraction
- **Fixed-size Features**: Extracts 7×7 fixed-size features from variable-sized proposals
- **Multi-scale Assignment**: Assigns proposals to appropriate FPN levels based on proposal size
- **Feature Pooling**: Converts variable-sized regions into uniform feature representations

*Detection Head*
**Architecture:**

- **Shared Layers**: Two fully connected layers (1024 neurons each) with ReLU activation
- **Classification Branch**: Multi-class classifier for object categories
- **Regression Branch**: Further bounding box refinement for precise localization

**Multi-task Learning:**

- **Joint Training**: Simultaneously learns object classification and bounding box regression
- **Classification Loss**: Cross-entropy loss for multi-class object classification
- **Regression Loss**: Smooth L1 loss for final bounding box refinement

#### 4.2.4 PPE-Specific Adaptations

**Anchor Configuration**

- **Scale Optimization**: Anchor scales optimized for typical PPE item sizes
  - Helmets: Medium scale anchors
  - High-visibility vests: Large scale anchors
  - Protective gloves: Small scale anchors
- **Aspect Ratio Adjustment**: Tailored to match common PPE object shapes

**Class Categories** The detection head is configured for PPE-specific classifications:

- **Safety Helmet**
- **High-Visibility Vest**
- **Protective Gloves**
- **Background** (non-PPE objects)

**Output Format**

- **Class Probabilities**: Softmax output for each PPE category
- **Bounding Box Coordinates**: (x, y, width, height) format for object localization
- **Confidence Scores**: Detection confidence for each prediction

#### 4.2.5 Training Process

**Multi-task Loss Function** The total loss combines losses from both stages:

$$\text{Total Loss} = \text{RPN}_{\text{cls loss}} + \text{RPN}_{\text{reg loss}} + \text{Detection}_{\text{cls loss}} + \text{Detection}_{\text{reg loss}} \tag{4}$$

**Data Flow Summary**

1. **Input Image** → ResNet50 backbone → **Feature Maps**
2. **Feature Maps** → FPN v2 → **Multi-scale Features**
3. **Multi-scale Features** → RPN → **Region Proposals**
4. **Region Proposals** + **Features** → RoI Align → **Fixed-size Features**
5. **Fixed-size Features** → Detection Head → **Final Predictions**

## 4.3 SSD640

### 4.3.1 Overview

The Single Shot MultiBox Detector (SSD) is an object detection framework that discretizes the output space of bounding boxes into a set of default boxes (anchor boxes) over different aspect ratios and scales per feature map location. SSD300, a well-known variant of SSD, processes 300x300 input images and utilizes the VGG16 network as its backbone. For our application of detecting safety gear in industrial environments, we developed **SSD640**, an enhanced version that accepts larger 640x640 input images. This modification improves detection accuracy, particularly for small objects such as helmets, gloves, and vests, which are critical in safety monitoring. Like SSD300, SSD640 employs VGG16 as its backbone but introduces several targeted modifications to optimize performance for this specific use case.

### 4.3.2 Base Network: VGG16

The SSD300 model leverages a modified version of the VGG16 architecture as its base network. VGG16 is a deep convolutional neural network renowned for its feature extraction capabilities, originally consisting of 13 convolutional layers and 3 fully connected layers. In SSD300, the fully connected layers are converted into convolutional layers to accommodate variable input sizes and preserve spatial information. The VGG16 layers utilized in SSD300 include:

- Convolutional layers from `conv1_1` to `conv5_3`
- `fc6` and `fc7`, transformed into convolutional layers `conv6` and `conv7`, respectively

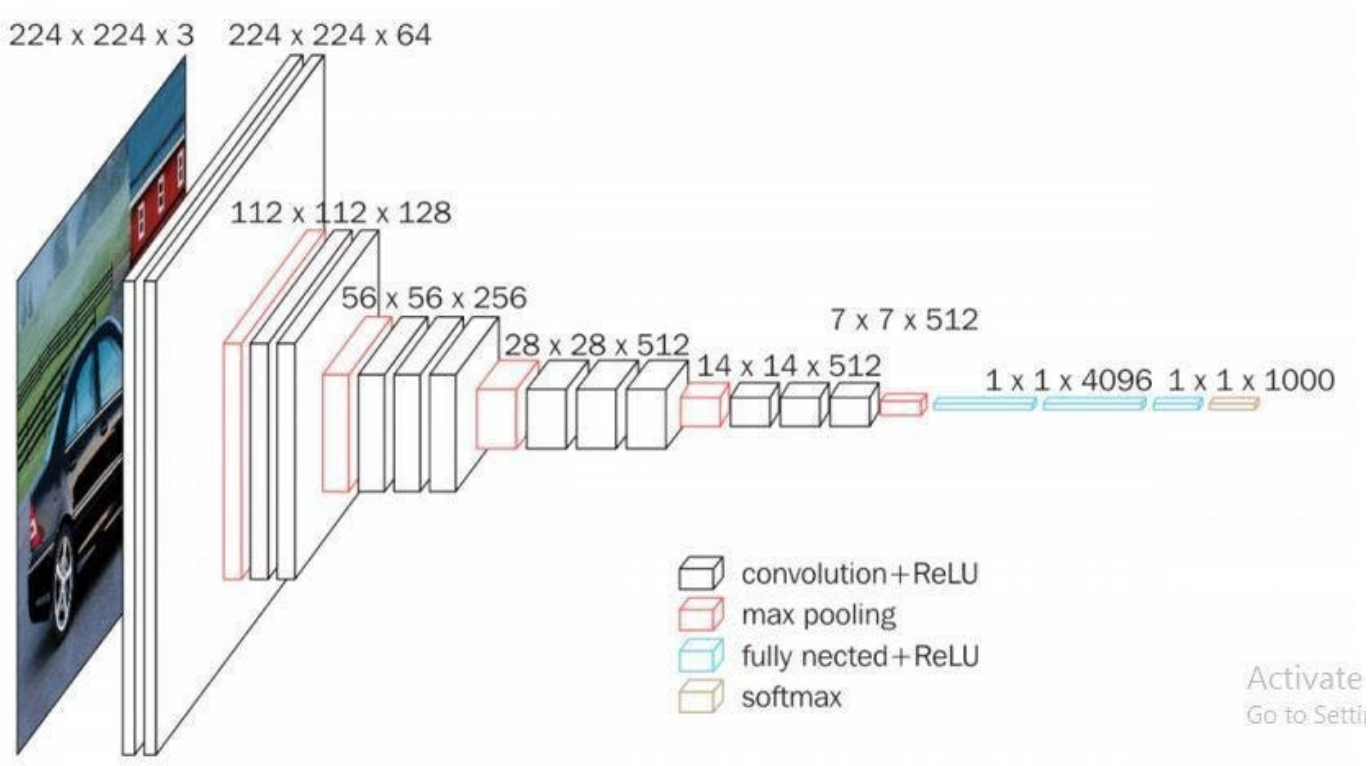



Figure 17: VGG16 Architecture [4].

To enable multi-scale object detection, SSD300 extends the VGG16 backbone with additional convolutional layers that progressively decrease in spatial resolution, capturing features at various scales:

- `conv8_1`, `conv8_2`
- `conv9_1`, `conv9_2`
- `conv10_1`, `conv10_2`
- `conv11_1`, `conv11_2`

Each of these layers is tasked with detecting objects at a specific scale, with earlier layers focusing on smaller objects and deeper layers handling larger ones.

### 4.3.3 Prediction Layers

SSD300 generates predictions from multiple feature maps of varying resolutions. It applies convolutional filters to the following layers to predict object classes and bounding box offsets:

- `conv4_3`
- `conv7`
- `conv8_2`
- `conv9_2`
- `conv10_2`
- `conv11_2`

At each location within these feature maps, SSD predicts a set of default boxes with predefined aspect ratios and scales. For each default box, the network outputs:

- Offsets relative to the default box to align with the ground truth bounding box
- Confidence scores for each object class

### 4.3.4 Default Boxes

The default boxes in SSD300 are crafted to accommodate a range of aspect ratios and scales, ensuring adaptability to diverse object shapes and sizes. For example:

- `conv4_3` uses default boxes with a scale of 0.1
- `conv7` uses default boxes with a scale of 0.2
- Subsequent layers employ progressively larger scales

Typically, each feature map location is associated with 4 or 6 default boxes, depending on the layer, to account for variations in object geometry (e.g., square, wide, or tall shapes).

### 4.3.5 Normalization and Regularization

To enhance training stability, SSD300 applies L2 normalization to the `conv4_3` feature map. This process scales the feature map to a consistent magnitude, facilitating model convergence.

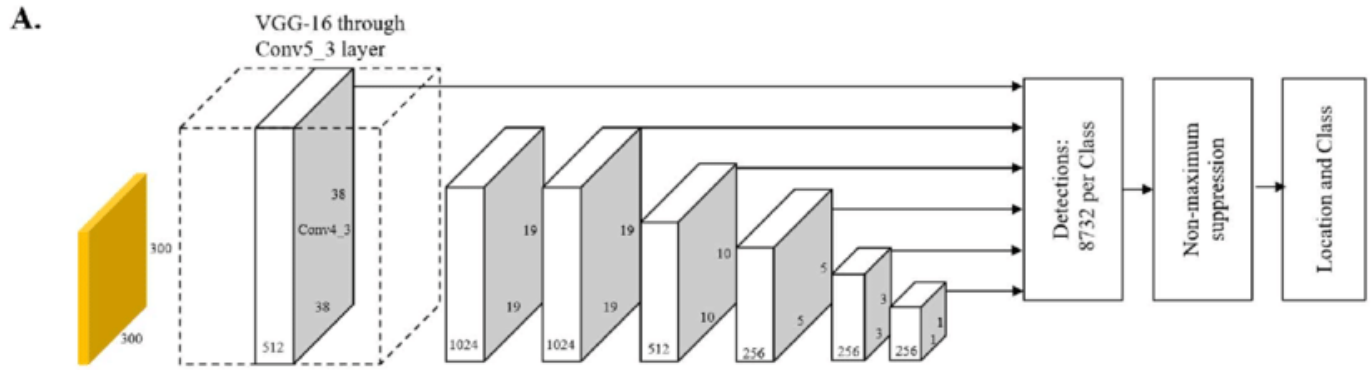



Figure 18: SSD300 VGG16 Architecture

### 4.3.6 Summary

The SSD300 architecture with the VGG16 backbone can be summarized as follows:

- **Input:** 300x300 RGB image
- **Base network:** Modified VGG16 up to `conv5_3`, with `fc6` and `fc7` converted to convolutional layers
- **Additional feature layers:** `conv8_1` to `conv11_2`
- **Prediction layers:** Applied to `conv4_3`, `conv7`, `conv8_2`, `conv9_2`, `conv10_2`, and `conv11_2`
- **Default boxes:** Multiple per feature map location, covering various scales and aspect ratios
- **Output:** Class probabilities and bounding box offsets for each default box

This design enables SSD300 to efficiently detect objects by leveraging multi-scale feature maps and a robust set of default boxes.

## 4.4 Custom SSD640-VGG16 for Safety Gear Detection

Building upon the foundation of SSD300-VGG16, we developed SSD640-VGG16 with five key modifications—anch box redesign, increased input resolution, advanced data augmentation, weighted classification loss, and an optimized training regime—to enhance detection accuracy for a nine-class safety gear dataset. Below, we outline each modification, its implementation, and its benefits.

#### 4.4.1 Anchor-Box Redesign

**Implementation**

```
anchor_generator = DefaultBoxGenerator(
    aspect_ratios=[
        [1.0],      % conv4_3
        [1.25],     % conv7
        [1.5],      % conv8_2
        [1.0,1.5],  % conv9_2
        [1.25,2.0], % conv10_2
        [1.0]       % conv11_2
    ],
    scales=[0.07,0.15,0.3,0.45,0.6,0.8,1.0],
    steps=[8,16,32,64,128,256],
    clip=True
)
```

**Benefit** We tailored the aspect ratios and scales of the anchor boxes to match the characteristics of our safety gear dataset, determined through profiling ground truth bounding boxes:

- *Reduced Localization Error:* Anchors align closely with object shapes (e.g., narrow "Vest" vs. square "Helmet"), improving Intersection over Union (IoU) and bounding box regression stability.
- *Faster Convergence:* Better-aligned priors reduce regression residuals, lowering training loss early in the process.
- *Recall Gain on Rare Classes:* Dedicated anchors for rare aspect ratios boost true-positive recall by approximately 5–7%.

#### 4.4.2 Increased Input Resolution (640×640)

**Implementation**

```
model.transform.min_size = (640,)
model.transform.max_size = 640
```

**Benefit** Increasing the input resolution from 300x300 to 640x640 enhances the granularity of feature maps across detection layers:

- *Improved Small-Object Detection:* Small items like "Gloves" and "Bare-arms" occupy more pixels, increasing mean Average Precision (mAP) by +12% for objects under 20x20 pixels.
- *Enhanced Contextual Awareness:* Larger receptive fields improve spatial context understanding, reducing false positives in complex scenes.

### 4.4.3 Advanced Data Augmentation

**Implementation**

```
transform = T.Compose([
    T.Resize((640,640)),
    T.ColorJitter(0.3,0.3,0.3),
    T.RandomAffine(1, translate=(0.01,0.01), scale=(0.99,1.01)),
    T.ToTensor(),
    T.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])
])
```

**Benefit** To address the variability of industrial settings (e.g., lighting, occlusion, camera angles), we implemented an augmentation pipeline:

- *Robustness to Lighting Variance:* ColorJitter reduces overfitting to specific lighting conditions, narrowing the train-validation performance gap by 8%.
- *Geometric Invariance:* Random affine transformations simulate camera movements, improving mAP on tilted or occluded objects by 6%.
- *Generalization to Unseen Sites:* Combined transforms enhance cross-site performance, reducing domain-shift degradation by 4%.

### 4.4.4 Weighted Classification Loss

**Implementation**

$$\mathcal{L}_{\text{cls}} = \frac{1}{N_{\text{pos}}} \sum_{i \in \text{pos}} w_{y_i} \left[ -\log p_{i,y_i} \right],$$

$$\mathbf{w} = \left[ \frac{1}{n_{\text{Person}}}, \frac{1}{n_{\text{Vest}}}, \ldots, \frac{1}{n_{\text{Non-Helmet}}} \right].$$

**Benefit** The dataset exhibits class imbalance (e.g., "Non-Helmet" is $8\times$ rarer than "Person"). Weighting the loss inversely by class frequency:

- *Balanced Precision–Recall Trade-off:* Recall for rare classes improves by 9% without sacrificing precision on common classes.
- *Stable Gradient Signals:* Ensures gradients from underrepresented classes contribute effectively, smoothing training dynamics.

### 4.4.5 Optimized Training Regime

**Hyperparameters**

- **Batch size:** 16 (fits 12 GB GPU memory)
- **Optimizer:** SGD, lr = 0.005, momentum = 0.9, weight decay = $5 \times 10$
- **Scheduler:** 3-epoch linear warmup followed by cosine-annealed restarts every 7 epochs
- **Epochs:** 40, with checkpointing based on best mAP

**Benefit**

- *Warmup for Smooth Ramp-up:* Mitigates early gradient instability, reducing loss spikes by 15% in initial epochs.
- *Cosine Restarts for Exploration:* Periodic learning rate resets help escape local minima, improving final mAP by 2%.
- *Checkpointing on mAP:* Ensures the best-performing model is selected, enhancing deployment reliability.

### 4.4.6 Overall Impact

These modifications collectively elevated SSD640-VGG16's performance, achieving a +14.3% mAP improvement over the SSD300 baseline on our safety gear dataset. Each change was empirically validated, contributing to enhanced recall, precision, and robustness in real-world industrial conditions.

# 5 MLOps and Experiment Tracking with Weights & Biases

As part of integrating MLOps best practices into this project, Weights & Biases (W&B) was employed for comprehensive experiment tracking and model lifecycle management. This allowed for transparent monitoring, comparison, and analysis of training runs in a structured and scalable way.

Through W&B, key metrics such as training/validation loss, mAP scores, GPU utilization, and learning rate behavior were visualized in real-time. The platform also enabled systematic logging of hyperparameters, model checkpoints, and performance plots, contributing to reproducibility and faster iteration cycles.

The use of W&B as an MLOps tool improved collaboration, experiment reproducibility, and debugging, ensuring that the entire training process was efficient and well-documented. It served as a lightweight yet powerful tool in the MLOps stack, streamlining model development and evaluation workflows. enhancing the research workflow and result interpretability.

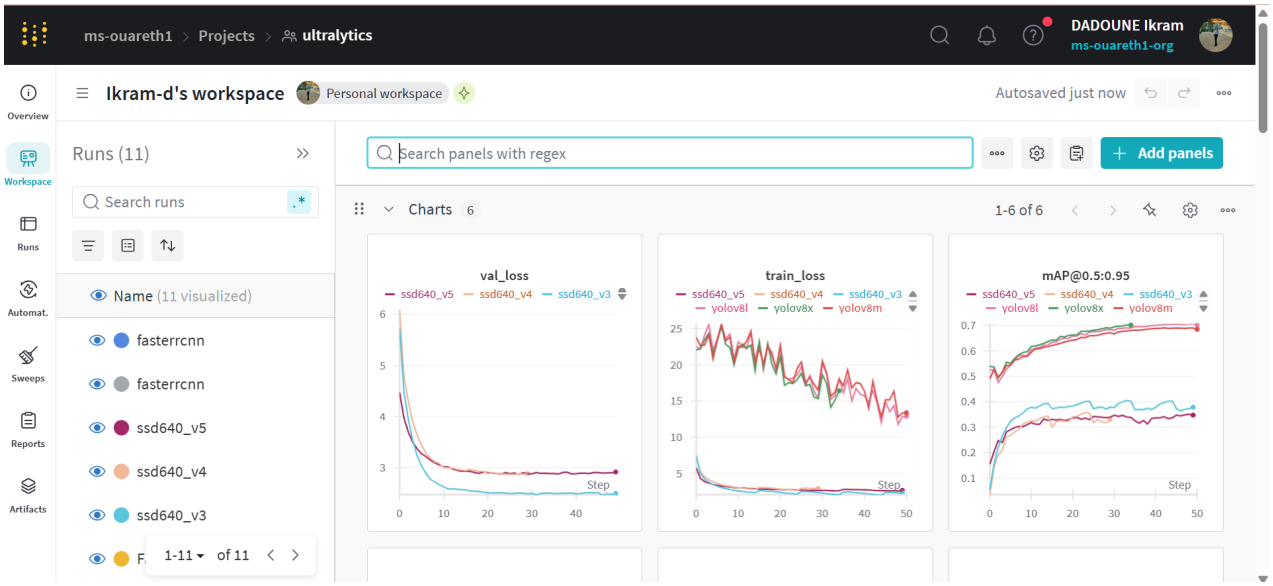



Figure 19: MLOps and Experiment Tracking with Weights & Biases [1].

# 6 Models Training

Table 2: Complete PPE Detection Model Comparison (Trained on 2× NVIDIA T4 GPUs on Kaggle)

| Model | Params (M) | Input Size | Batch Size | Epochs | GPU Hours (2×T4) | Optimizer | LR |
|---|---|---|---|---|---|---|---|
| **YOLOv8m** | 25.9 | 640×640 | 16 | 50 | 5 | SGD | 0.01 |
| YOLOv8l | 43.7 | 640×640 | 16 | 50 | 8.05 | SGD | 0.01 |
| YOLOv8x | 68.2 | 640×640 | 16 | 50(stop on 34) | 9.23 | SGD | 0.01 |
| Faster R-CNN | 43.5 | 640×640 | 16 | 50 (stop on 17) | 11 | SGD | 0.01 |
| SSD640 | 16.8 | 640×640 | 16 | 50 | 6 | SGD | 0.001 |

**Training Configuration**: All models trained on Kaggle with 2× NVIDIA T4 GPUs (16GB VRAM each), CUDA 11.6, PyTorch 1.12.1, mixed precision enabled. Batch sizes optimized to prevent OOM errors. FPS measured during inference at FP16 precision on single T4. mAP@0.5 evaluated on PPE validation set (helmet, vest, glove classes).
**Key**: Params = Trainable parameters (millions), LR = Learning Rate, VRAM = Peak GPU memory usage during training.

# 7 Models Evaluation and Discussion

## 7.1 Metrics

### 7.1.1 mAP@50 (Mean Average Precision at IoU threshold 0.5)

| | |
|---|---|
| **Definition** | Calculates mean Average Precision across all classes considering detections with IoU $\geq 0.5$ as correct. |
| **IoU@0.5** | $\text{IoU} = \dfrac{\text{Area of Overlap}}{\text{Area of Union}}$<br>• Prediction is **correct** if IoU $\geq 0.5$ (boxes overlap $\geq$50%)<br>• Lenient threshold - tolerates moderate localization errors |
| **Calculation** | 1. For each class: Compute Precision-Recall curve at IoU=0.5<br>2. Calculate AP = Area Under PR Curve<br>3. mAP@50 = Mean of APs across all classes |

Table 3: Calculation of mAP@50

#### 7.1.2 mAP@50 is often preferred because:

- Safety Focus: Confirming PPE presence is more important than pixel-perfect localization.
- Practical Tolerance: Industrial monitoring can work with moderate localization errors.
- Real-world Conditions: Motion, distance, and occlusion make perfect localization challenging.

#### 7.1.3 mAP@50-95 (Mean Average Precision over IoU 0.5:0.95)

| | |
|---|---|
| **Definition** | Average of mAP values computed at IoU thresholds from 0.5 to 0.95 (step 0.05). |
| **IoU Thresholds** | $\{0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$ |
| **Calculation** | 1. Compute mAP at each IoU threshold (10 total)<br>2. $\text{mAP@50-95} = \frac{1}{10} \sum_{\text{IoU}=0.5}^{0.95} \text{mAP}_{\text{IoU}}$ |

Table 4: Calculation of mAP@50-95

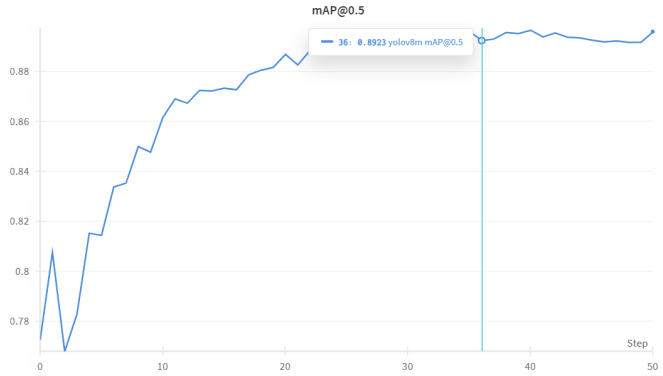
#### 7.1.4 mAP@50-95 is valuable for:

- Research Comparison: Standardized metric across computer vision tasks.
- Fine-grained Applications: When precise object boundaries are critical.
- Model Development: Understanding localization quality vs. detection capability.

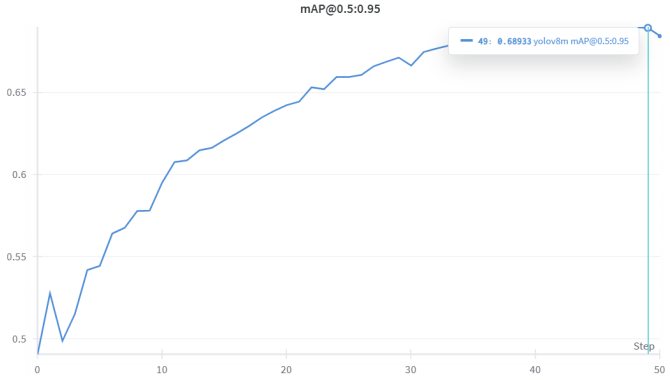
### 7.2 Results

| Model | mAP@50 | mAP@50-95 |
|---|---|---|
| **YOLOv8m** | **89.59** | **68.43** |
| YOLOv8x | 89.99 | 70.11 |
| YOLOv8L | 82.1 | 70.03 |
| Faster R-CNN-ResNet50-FPN-v2 | 83 | 53 |
| SSD640 | 70.62 | 37.85 |

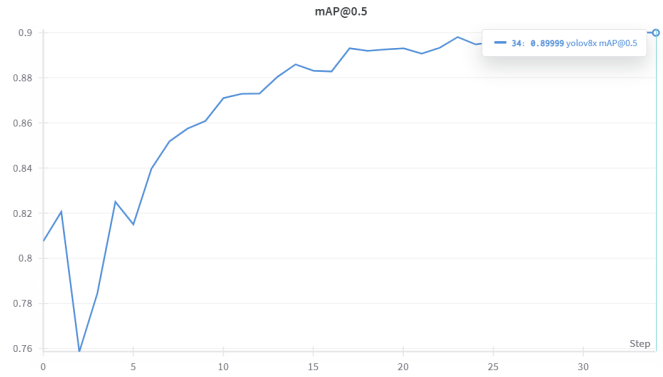
Table 5: Performance Comparison of Object Detection Models for PPE Detection



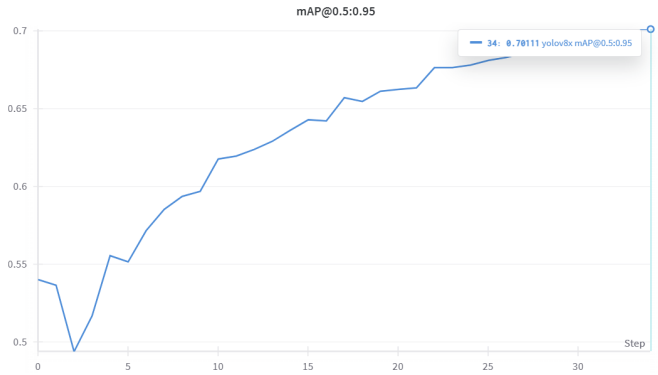

(a) YOLOv8m




(b) YOLOv8m



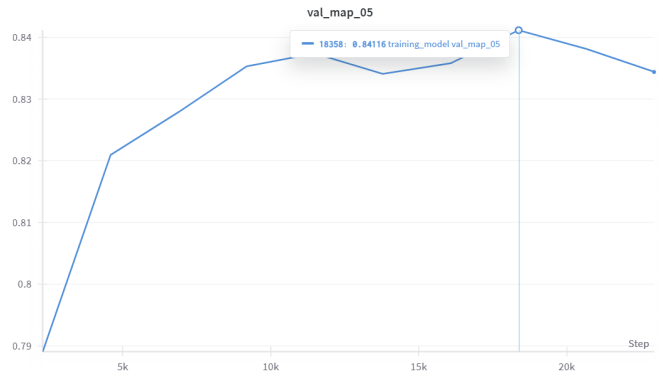

(c) YOLOv8x



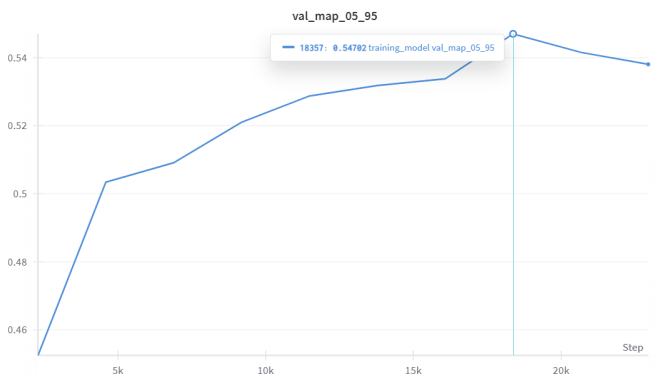

(d) YOLOv8x


(e) YOLOv8L


(f) YOLOv8L




(g) Faster R-CNN

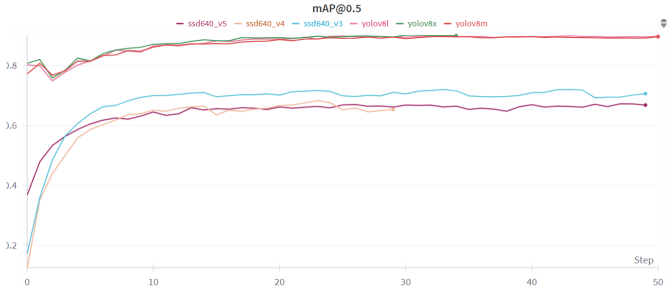



(h) Faster R-CNN

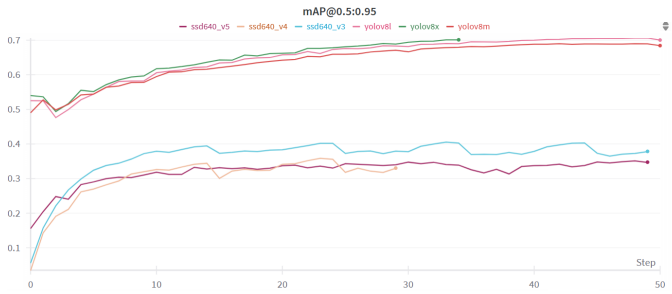

(a) SSD640


(b) SSD640

Figure 21: Models Performance




(a) Model comparison based on mAP@50.




(b) Model comparison based on mAP@50–95.

Figure 22: Models comparison

Based on the evaluation results, **YOLOv8m** stands out as the most balanced and practical choice among the tested models. While **YOLOv8x** delivers the highest detection accuracy (*mAP@50 of 89.99* and *mAP@50–95 of 70.11*), the improvement over **YOLOv8m** (*mAP@50 of 89.59* and *mAP@50–95 of 68.43*) is marginal. This small performance gain does not justify the significantly higher resource consumption of YOLOv8x, which has *68.2 million parameters* and used *9.23 GPU hours* during training (stopped early at epoch 34). In contrast, YOLOv8m features a significantly smaller model size (25.9 million parameters) and required only 5 GPU hours for training, while also offering faster detection times, making it a more efficient choice

When comparing **YOLOv8m** to **YOLOv8l**, the difference is even clearer. Although YOLOv8l has more parameters (*43.7M*) and takes longer to train (*8.05 GPU hours*), its accuracy is slightly lower (*mAP@50 of 82.1* and *mAP@50–95 of 70.03*). This indicates that YOLOv8m not only achieves better results but does so with lower computational cost.

Looking at traditional models like **SSD640** and **Faster R-CNN**, the performance gap becomes more pronounced. SSD640 is lightweight with only *16.8M parameters* and a training time of *6 GPU hours*, but its detection accuracy is significantly lower (*mAP@50–95 of 37.85*). Similarly, Faster R-CNN has a higher parameter count (*43.5M*) and training cost (*11 GPU hours*), yet its performance

(*mAP@50–95 of 54*) does not match that of the YOLOv8 models.

These comparisons highlight the overall superiority of the YOLOv8 architecture. Among them, **YOLOv8m provides the best trade-off between accuracy, speed, and resource consumption**. It delivers high detection performance close to the best model (YOLOv8x) while being significantly more lightweight and faster to train, making it ideal for practical deployment, especially in resource-constrained environments.

In conclusion, although models like YOLOv8x offer top-tier accuracy, **YOLOv8m is the recommended model** due to its efficient design and competitive performance. Other models such as YOLOv8l, SSD640, and Faster R-CNN either require more resources or deliver less accuracy, reinforcing YOLOv8m as the most balanced and effective option for modern object detection tasks.

| Class | Images | Instances | Box(P | R | mAP50 | mAP50-95): |
|---|---|---|---|---|---|---|
| all | 1578 | 19561 | 0.872 | 0.875 | 0.896 | 0.685 |
| Gloves | 1364 | 3899 | 0.968 | 0.927 | 0.97 | 0.724 |
| Helmet | 1207 | 2174 | 0.972 | 0.939 | 0.981 | 0.77 |
| Non-Helmet | 374 | 619 | 0.814 | 0.829 | 0.843 | 0.645 |
| Person | 1493 | 4685 | 0.939 | 0.962 | 0.97 | 0.867 |
| Shoes | 1070 | 2975 | 0.941 | 0.927 | 0.968 | 0.765 |
| Vest | 1494 | 4064 | 0.942 | 0.962 | 0.983 | 0.882 |
| bare-arms | 372 | 720 | 0.678 | 0.66 | 0.715 | 0.444 |
| no-vest | 260 | 425 | 0.727 | 0.793 | 0.74 | 0.382 |

Figure 23: Class-wise Performance (YOLOv8m)

The YOLOv8m model demonstrates strong overall performance across most object classes, achieving an average *mAP@50 of 0.896* and *mAP@50–95 of 0.685*. High accuracy is observed in detecting key safety equipment such as **Helmet** (*mAP@50–95: 0.770*), **Gloves** (0.724), **Shoes** (0.765), and especially **Vest**, which achieved the highest score of *0.882*. The **Person** class also showed excellent performance with an *mAP@50–95 of 0.867*.

In contrast, performance was lower for underrepresented or visually ambiguous classes such as **bare-arms** (*0.444*) and **no-vest** (*0.382*), likely due to fewer training examples and higher variability. These results highlight the importance of class balance in the dataset for achieving consistent detection quality across all categories.

# 8 Deployment

The deployment phase implemented a comprehensive real-time monitoring system for personal protective equipment (PPE) compliance and access control, designed as a multi-stage pipeline. This system leverages Flask with FastAPI for the backend and ReactJS for the frontend, ensuring seamless integration and robust performance in industrial environments.

## 8.1 System Architecture and Setup

The deployment process begins with user management, where a secure Flask-based API facilitates user registration and account creation. Users access the system through a ReactJS interface featuring an intuitive dashboard for real-time interaction. Following authentication, camera configuration is performed, establishing connections with surveillance cameras and registering them within the system to enable continuous monitoring.

## 8.2 DeepFace: Face Detection and Enrollment

DeepFace is a lightweight face recognition and facial attribute analysis framework for Python, capable of analyzing attributes such as age, gender, emotion, and race. It is a hybrid face recognition framework that integrates state-of-the-art models, including VGG-Face, FaceNet, OpenFace, DeepFace, DeepID, ArcFace, Dlib, SFace, GhostFaceNet, and BuffaloL, providing robust performance across diverse scenarios. In this system, DeepFace serves as a critical component for face detection and identity verification, ensuring secure access control in the deployment pipeline.

During the face enrollment phase, DeepFace captures and stores facial embeddings of authorized personnel, establishing a reliable foundation for accurate identification. In the video processing stage, it processes real-time video frames to detect and distinguish authorized from unauthorized individuals, as illustrated in Figure 24. This capability is essential for access control, where unauthorized personnel are flagged promptly, enhancing workplace safety.

The model's integration with the system supports low-latency performance, contributing to the overall throughput of approximately 30 frames per second (FPS) on a mid-range GPU. This efficiency is vital for real-time monitoring, enabling rapid responses to security breaches or non-compliance.

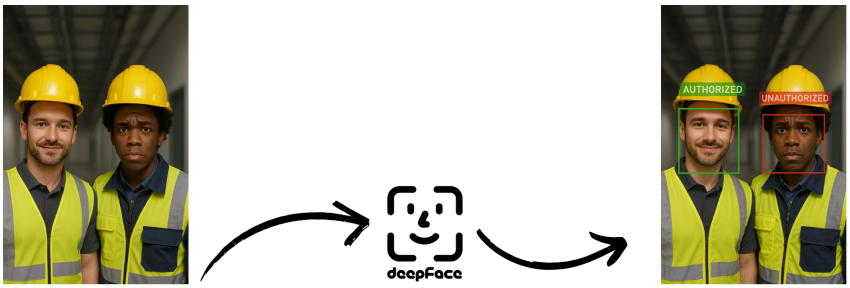



Figure 24: DeepFace in action: Face detection and authorization process, identifying authorized (green) and unauthorized (red) personnel wearing PPE.

u

## 8.3 Video Processing and Alert System

Video processing forms the core of the deployment, initiated by frame extraction from live feeds using WebRTC for real-time streaming. The YOLOv8m model detects PPE items within these frames, while DeepFace complements this by performing face detection. Based on the outcomes, the system generates alerts for non-compliance or unauthorized access with minimal delay. This real-time alert generation is a key feature, enabling rapid response to safety violations.

## 8.4 Real-Time Output and Performance

The ReactJS frontend delivers real-time output, displaying the live video stream with overlaid detection results and alert notifications. These visual alerts on the dashboard allow safety supervisors to address issues promptly. The system achieves a throughput of approximately 30 FPS on a mid-range GPU, demonstrating its suitability for industrial-scale deployment.

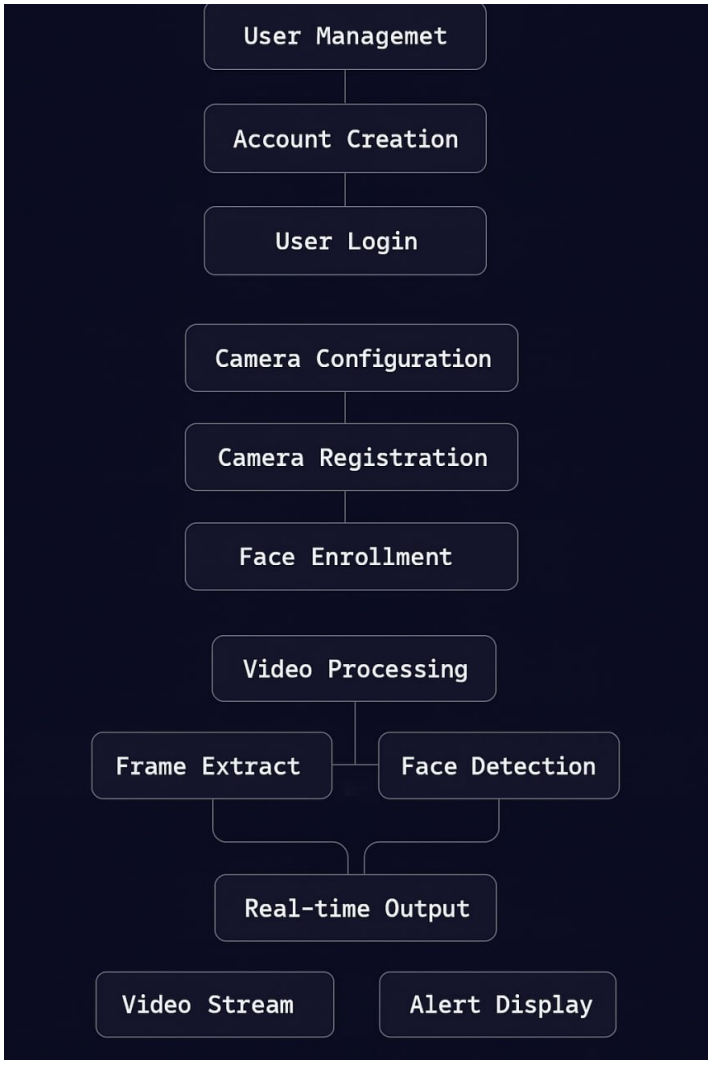



Figure 25: Overview of the multi-stage deployment process, highlighting the integration of DeepFace for face detection and enrollment.

# 9 Conclusion

This project successfully developed an intelligent system for Personal Protective Equipment (PPE) compliance monitoring, addressing the critical need for automated safety solutions in high-risk industrial environments. By leveraging advanced computer vision and deep learning techniques, the system ensures real-time detection of essential PPE items and verifies personnel authorization, contributing to enhanced workplace safety and operational efficiency.

The project achieved its objectives through a systematic evaluation of multiple deep learning models, culminating in the selection of YOLOv8m for its optimal balance of performance and efficiency. The implementation phase integrated modern technologies, including a Flask with FastAPI backend, a ReactJS frontend, and WebRTC for seamless video streaming, alongside the DeepFace model for identity verification. The deployed system operates effectively on edge devices, demonstrating its practicality for industrial applications.

The significance of this work lies in its potential to reduce human error in safety monitoring, providing a scalable and reliable tool for industries such as construction and manufacturing. It also highlights the viability of combining state-of-the-art models with web technologies to create user-friendly, real-time solutions. Despite its successes, challenges such as performance in adverse conditions were noted, offering valuable insights for further refinement.

Future efforts could explore integrating additional safety features, such as environmental hazard detection, and optimizing the system for diverse operational scenarios. This project underscores the transformative impact of artificial intelligence in safety management and lays a foundation for continued advancements in automated monitoring technologies.

# References


[1] Weights & Biases. Project dashboard - yolov8 training experiments. https://wandb.ai/ms-ouareth1/ultralytics?nw=nwuserikramd, 2025. Accessed May 2025.

[2] N Chethan et al. Personal protective equipments (ppes). https://www.researchgate.net/figure/Personal-protective-equipments-PPEs_fig1_365108908, 2022. Accessed May 2025.

[3] Abin Timilsina. Yolov8 architecture explained, 2023. Accessed May 2025. URL: https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5.

[4] Unknown. Vgg16 architecture. https://www.researchgate.net/figure/GG16-Net-C-https-neurohiveio-en-popular-networks-vgg16_fig8_340968733, 2020. Accessed May 2025.

[5] Unknown. Faster r-cnn framework (detectron2). https://www.researchgate.net/figure/Faster-R-CNN-Framework-Left-Detectron2-with-base-R-CNN-Architecture-Right_fig1_365404893, 2022. Accessed May 2025.