Available online at www.HighTechJournal.org

HighTech and Innovation Journal

ISSN: 2723-9535

Vol. 6, No. 3, September, 2025

Efficient Object Detection with an Optimized YOLOv8x Model

Nafiz Fahad [1], Md. Jakir Hossen [2, 3*], Md Shohel Sayeed [4]

[1] Faculty of Information Science and Technology (FIST), Multimedia University, Melaka, 75450, Malaysia.

[2] Center for Advanced Analytics (CAA), CoE for Artificial Intelligence, Faculty of Engineering & Technology (FET), Melaka, 75450, Malaysia.

[3] ELITE Research Lab, 17010 Cedarcroft Rd, Queens, New York, United States.

[4] Centre for Intelligent Cloud Computing, CoE for Advanced Cloud, Faculty of Information Science & Technology, Multimedia University, Melaka, 75450, Malaysia.

Received 23 February 2025; Revised 09 August 2025; Accepted 17 August 2025; Published 01 September 2025

Abstract

This study developed an efficient object detection model for indoor environments, addressing common challenges such as occlusions, varying lighting, and cluttered scenes. We evaluated several YOLOv8 variants—ranging from nano to extra-large—and introduced an optimized YOLOv8x model. Our approach combines structured pruning, quantization-aware training, and advanced data preprocessing techniques, including augmentation and noise reduction, to improve model performance while reducing computational demands. The models were developed and evaluated using a carefully selected indoor object detection dataset featuring ten common categories. Performance was measured through key metrics like precision, recall, and mean average precision (mAP). Among them, the fine-tuned YOLOv8x clearly outshined the baseline models, reaching a training precision of 0.577, a recall of 0.572, and an mAP@0.50 of 0.537. When tested on new data, it demonstrated even better generalization, delivering a precision of 0.502, a recall of 0.528, and an mAP@0.50 of 0.480—proving robustness and reliability in real-world scenarios. These results demonstrate that pruning and quantization can significantly reduce model complexity without sacrificing accuracy, which helps to detect indoor objects. In essence, it is optimized for indoor object detection, offering promising applications in smart environments, surveillance, and robotics.

Keywords: Indoor Object; Object Detection; Computer Vision; YOLOv8; Deep Learning.

1. Introduction

For computer vision, object detection is an important part, which involves identifying and recognizing various objects within an image or complex scene. The ability to simultaneously detect, classify, and locate multiple objects makes this task essential for a vast scope of practical applications [1]. Despite being one of the most important challenges in computer vision and image analysis, recent advancements have significantly improved performance, largely due to enhanced methods for representing objects and the adoption of deep neural networks [2].

Moreover, object detection plays a crucial role across a wide range of industries, including autonomous driving, security monitoring, medical imaging, retail automation, precision farming, and quality control in manufacturing. Thanks to rapid advancements in deep learning and AI, the accuracy and efficiency of object detection keep improving, making it a cornerstone of future technological innovations. This topic has captured significant attention from both researchers and industry leaders alike. At its core, object detection focuses on creating computer models that enable vision-based systems to accurately interpret and understand their surroundings. Because of its importance and ongoing progress, object detection remains a vibrant and rapidly evolving field of research and practical application [3-5].

* Corresponding author: jakir.hossen@mmu.edu.my

http://dx.doi.org/10.28991/HIJ-2025-06-03-09

Therefore, traditional navigation aids, such as white canes, offer limited situational awareness and lack the capability to detect complex obstacles within dynamic and cluttered environments (Khan et al. [6]). Artificial intelligence and computer vision have become powerful tools in overcoming many complex challenges through advanced object detection techniques. Modern deep learning models—especially those from the YOLO (You Only Look Once) series—have set new standards by striking an effective balance between speed and accuracy in real-time detection tasks (Ben Atitallah et al. [7]). For instance, Feng & Jin [8] boosted the YOLOv8 model with cutting-edge components to enhance its ability to detect objects accurately in tough underwater conditions. Beyond that, numerous studies have explored deep learning's potential in object detection for Autonomous Mobile Robots (AMRs). Baheti et al., for example, designed a convolutional neural network to identify distracted drivers by recognizing behaviors like talking, sleeping, or eating through face and hand tracking [9]. In warehouse settings, another research team employed the MobileNet-SSD architecture to detect damage in pallet racks, proving the model's practical value in inspection tasks [10]. Similarly, Li et al. developed a MobileNet-SSD-based method for surface defect detection that fine-tunes the network to deliver real-time accuracy suitable for industrial production lines [11].

More recently, Tian et al. utilized the YOLOv12 model for object detection [12], while Zhou demonstrated real-time capabilities using the YOLO-NL detector [13]. Building on these advances, Wang et al. introduced a groundbreaking approach that eliminates the need for Non-Maximum Suppression (NMS) by employing a consistent dual assignment strategy during model training—streamlining and improving detection performance. Their design features two parallel heads: one uses a one-to-many assignment to provide rich supervision, and the other employs a one-to-one assignment to enable efficient inference without NMS. Both heads share a unified matching metric, ensuring harmonious training. Additionally, the model architecture is revamped to balance efficiency and accuracy—incorporating a lightweight classification head to cut computational costs, spatial-channel decoupled downsampling to optimize feature extraction, and a rank-guided block design that dynamically adjusts complexity based on layer redundancy. For further accuracy gains, YOLOv10 integrates large-kernel convolutions in smaller models and a partial self-attention module that captures global context efficiently with minimal overhead. Together, these advancements empower YOLOv10 to deliver fast, precise, and truly end-to-end object detection [14]. Additionally, Pulipalupula et al. used the YOLOv3 algorithm for object detection, highlighting object detection efficiency compared to RCNN, Fast RCNN, and SSD. YOLO achieves higher accuracy and faster processing by detecting objects in a single pass using CNNs. It shows promising results for applications like autonomous driving and security, with future scope in weapon detection [15]. Dang et al. presents a comprehensive benchmark of 25 YOLO object detectors for multi-class weed detection in cotton fields. It introduces a large, diverse dataset collected under natural conditions, evaluates model accuracy and speed, and demonstrates YOLO's effectiveness for real-time weed identification, aiding sustainable precision agriculture advancements [16].

Lee & Hawng (2022) improved YOLO's real-time object recognition on embedded systems with limited resources by using an adaptive frame control (AFC) technique. By dynamically controlling input frames and reducing latency without changing YOLO's core, AFC preserves detection accuracy. AFC enables cost-effective deployment by enhancing real-time performance on a variety of hardware, according to experiments [17]. Zhang et al. use a weighted BiFPN for enhanced multi-scale feature fusion and include multi-head self-attention into YOLO's CSP-Darknet backbone. With robust, context-aware feature extraction, their transformer-based ViT-YOLO outperforms previous techniques on VisDrone benchmarks, improving identification accuracy, particularly for small objects in complicated drone images [18]. Furthermore, Aulia et al. introduced a CNN-based object detection system for AMRs that employs real-world vehicle datasets, enhancing the robots' ability to identify various objects in dynamic environments [19].

However, in this current research we proposed a new optimized yolov8x model for AMR. Additionally, the contribution of the current study is mentioned below-

- Yolov8n (yolov8 nano), YOLOv8s (small), Yolov8m (medium), Yolov8l (large), and Yolov8x (extra-large) models used by this study.
- This study applied data preprocessing, augmentation, normalization, noise reduction, and feature extraction to balance the dataset
- We proposed an optimize yolov8x based object detection model to remove overfitting and underfitting.

This paper is organized as follows: The suggested technique is described in Section 2, the results and analysis are discussed in Section 3, additional details are given in Section 4, the limitations are discussed in Section 5, future work is examined in Section 6, and the conclusion is given in Section 7.

2. Proposed Method

This section provides a detailed description of the YOLOv8 model architecture, along with the procedures used for dataset preparation, training, optimization, validation, and testing. Figure 1 illustrates the overall workflow of the proposed approach. The dataset employed in this study originates from the Obstacles Detection for Blind People project and is adapted from an existing obstacle detection dataset available on GitHub [20, 21]. It comprises ten object

categories: door, opened door, cabinet door, refrigerator door, window, chair, table, cabinet, sofa/couch, and pole. The dataset consists of 1,012 training images and 107 testing images [22]. Table 1 provides a comprehensive summary of the dataset, detailing the distribution of samples across object categories such as doors, cabinet doors, refrigerator doors, windows, and chairs, as well as their allocation within the training, validation, and test subsets. Additionally, a YAML configuration file was developed, and directory structures were established for the training, validation, and test datasets. The architecture of the YOLOv8 model is depicted in Figure 2.

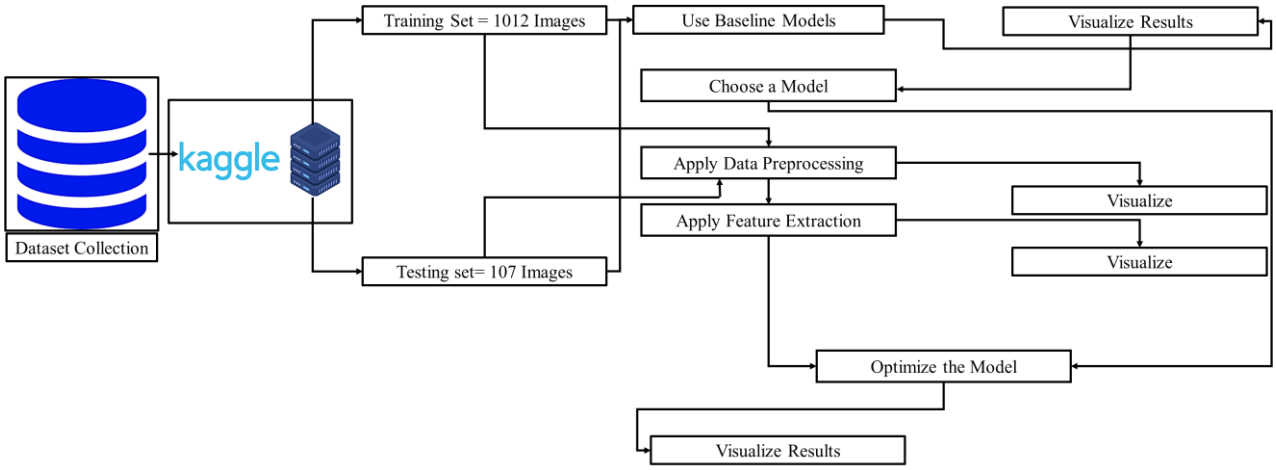


Figure 1. Workflow Diagram of this proposed method

Table 1. Class Distribution and Data Volume for Train, Validation, and Test Sets [16]

Mode	Door	cabinetDoor	refrigeratorDoor	window	chair	table	cabinet	couch	openedDoor	pole	Data_Volume
train	309	448	332	187	105	165	138	22	68	14	1008
test	25	42	1	49	38	41	44	31	18	3	104

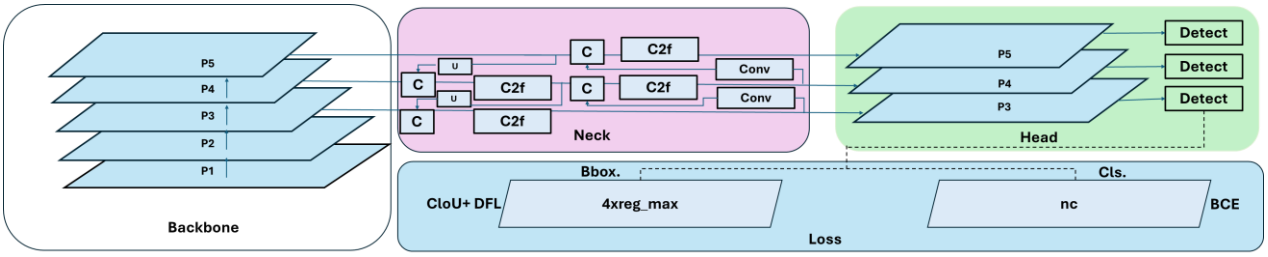


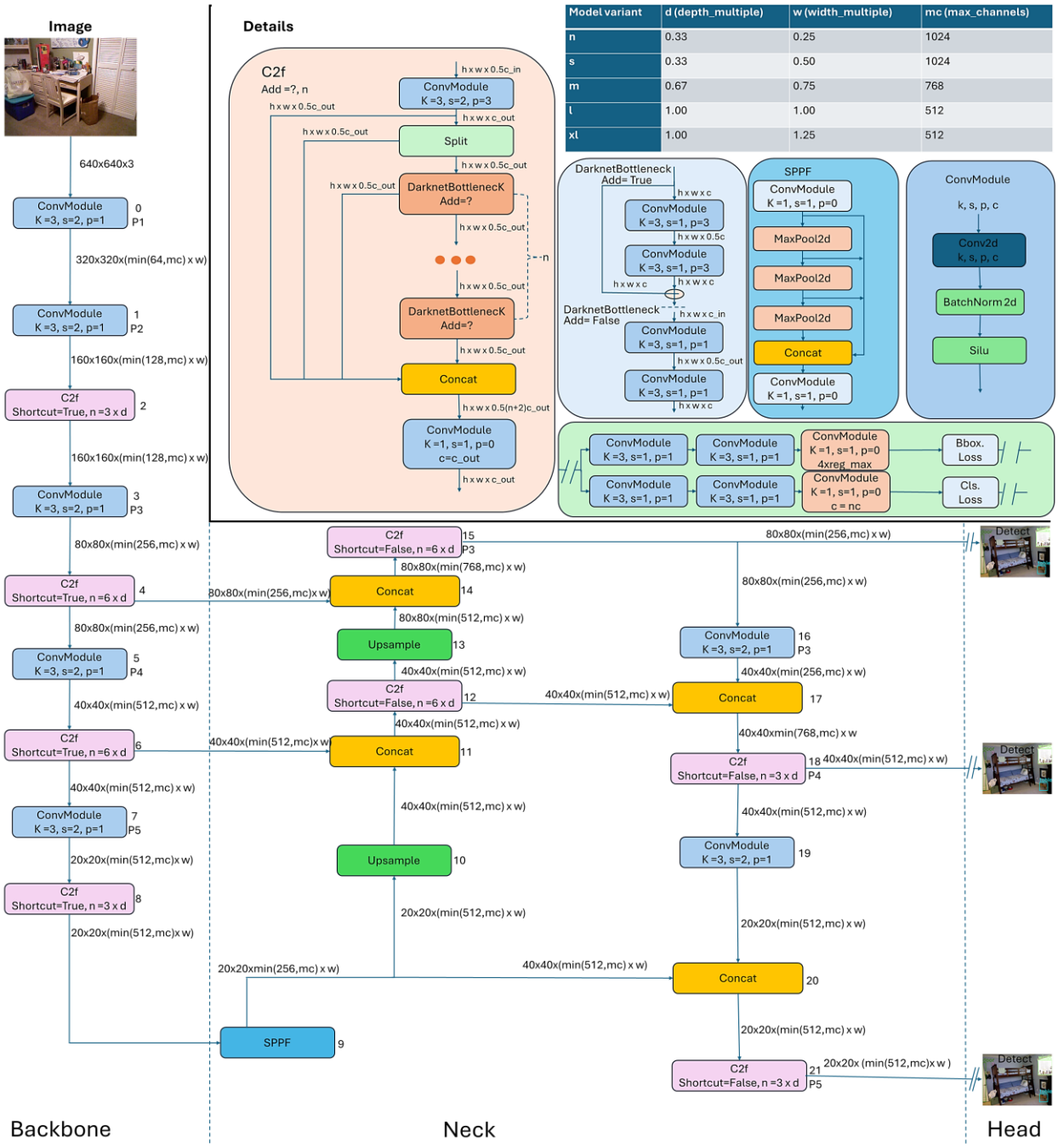
Figure 2. Four parts of the YOLOv8 algorithm's architecture: head, neck, loss, and backbone

2.1. Yolov8 Model Architecture

The head, neck, and backbone make up our model architecture, as seen in Figure 2. The design concepts for each component and the modules are shown in the following subsections.

2.1.1. Backbone

The model's backbone employs the Cross Stage Partial (CSP) architecture [23], which works by splitting the feature map into two halves. One half undergoes convolutional processing, while the other half bypasses this step and is later concatenated with the processed output. This clever design boosts the learning efficiency of convolutional neural networks (CNNs) while simultaneously reducing computational costs. Building on this, YOLOv8 introduces the C2f module, which merges the ELAN structure from YOLOv7 with the traditional C3 module to enable the network to capture more detailed gradient flow information [24, 25]. As shown in Figure 3, the C2f module consists of two ConvModules and several DarknetBottleNecks, linked through splitting and concatenation operations. In comparison, the older C3 module contains three ConvModules and multiple DarknetBottleNecks. Each ConvModule includes convolution, batch normalization, and SiLU activation layers, with 'n' representing the number of bottleneck blocks. Unlike YOLOv5.53, which uses the C3 module, this model opts for the more efficient C2f. Additionally, the number of blocks per stage has been trimmed down relative to YOLOv5—specifically, Stages 1 through 4 now have 3,6, 6, and 3 blocks respectively—to ease computational demands. Lastly, the traditional Spatial Pyramid Pooling (SPP) in Stage 4 is swapped out for the faster and more effective Spatial Pyramid Pooling - Fast (SPPF) module [26], enhancing both the model's learning capability and its inference speed.



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

Figure 3. YOLOv8 model architecture in detail

2.1.2. Neck

In general, deeper networks are better in dense prediction because they capture more feature information. Excessive convolution operations, especially for small objects, may result in information loss, and overly deep networks can reduce object location accuracy. The challenge of multi-scale feature fusion is tackled through the integration of the Path Aggregation Network (PAN) architecture [27] along with the Feature Pyramid Network (FPN) [28]. Figure 3 illustrates how the Neck component of our model combines data from many layers, enabling the upper layers to obtain more feature information from additional layers while the bottom layers, with fewer convolutional layers, maintain more accurate position information. Motivated by YOLOv5, where the lower feature map is strengthened by FPN up sampling and the upper feature map is strengthened by PAN down sampling, we combine these outputs to produce precise predictions for a range of image sizes. In our model, we use the FP-PAN (Feature Pyramid-Path Aggregation Network), which reduces processing needs by skipping convolution operations during up sampling.

2.1.3. Head

The proposed model departs from the YOLOv5 design by incorporating a decoupled head architecture [29], which separates the classification and detection components. As illustrated in Figure 3, the objectness branch has been removed,

retaining only the classification and regression branches. While the traditional Anchor-Based approach utilizes multiple anchors within an image to calculate the four positional offsets between anchors and objects for precise localization, the present model adopts an Anchor-Free strategy [30-33]. This method identifies the center of the object and determines the distances from this central point to the bounding box edges, enabling accurate object localization without relying on predefined anchors.

2.1.4. Loss

To train our model, we assign positive and negative samples using the Task Aligned Assigner from Task-aligned One-stage Object Detection (TOOD). Equation 1 shows how this method evaluates weighted scores from both regression and classification in order to choose positive samples.

$$t = s^{\alpha} \times u^{\beta} \tag{1}$$

Here, s represents the predicted score for the labeled class, while u denotes the IoU between the predicted bounding box and the ground truth bounding box. Furthermore, the model incorporates distinct branches for classification and regression tasks. The classification branch employs Binary Cross-Entropy (BCE) Loss, defined as follows:

$$Loss_n = -w\ [y_n \log x_n + (1 - y_n) \log (1 - x_n)] \tag{2}$$

where w is the weight, y_n represents the labeled value, and x_n is the model's predicted value.

Moreover, the regression component employs a combination of Distribution Focal Loss (DFL) and Complete Intersection over Union (CIoU) Loss. The DFL approach enhances the accuracy of probability predictions near the object's target value, represented as y, which is defined by:

$$DFL\ (S_n, S_{n+1}) = -((y_{n+1} - y) \log(S_n) + (y - y_n) \log(S_{n+1})) \tag{3}$$

with S_n and S_{n+1} defined as:

$$S_n = \frac{(y_n{}^{+1} - y)}{(y_n{}^{+1} - y_n)}\ S_n{}^{+1} = \frac{(y - y_n)}{(y_n{}^{+1} - y_n)} \tag{4}$$

Additionally, CIoU Loss enhances Distance IoU (DIoU) Loss by incorporating a factor that accounts for the difference in aspect ratios between the predicted and the actual bounding boxes. This relationship is expressed as:

$$CIoU_{\text{loss}} = 1 - IoU + \left(\frac{Distance^2}{Distance_C^2}\right) + \left(\frac{v^2}{(1 - IoU + v)}\right) \tag{5}$$

where ν is a parameter measuring aspect ratio consistency, calculated as:

$$v = \frac{4}{\pi^2} \left(\arctan\left(\frac{w^{gt}}{h^{gt}}\right) - arctan\left(\frac{w^{p}}{h^{p}}\right)\right)^2 \tag{6}$$

2.2. Model Setup

The YOLOv8 models, including YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large)—were all run on a T4 GPU, with every model's parameters set to be trainable. In addition, the YOLOv8x model was optimized further. Table 2 compares these models in detail, looking at how complex and efficient each one is. It shows three important factors: total parameters, trainable parameters, and FLOPs (floating point operations per second), which together affect the model's size, how fast it trains, and how quickly it makes predictions. The models vary greatly—from the smallest YOLOv8n with 3.2 million parameters and 8.7 GFLOPs to the largest YOLOv8x with 68.1 million parameters and 275 GFLOPs—giving options depending on the needs for accuracy and computing power. The optimized YOLOv8x reduces complexity by pruning away unnecessary parameters and using quantization to convert calculations to lower precision. This cuts down the number of parameters by about 30%, from 68.1 million to 47 million, and lowers FLOPs from 275 to 190 GFLOPs. These changes make the model faster and use less memory, without losing accuracy. Because of this, the optimized YOLOv8x is well-suited for real-time tasks like autonomous robots, vehicles, security systems, and drone vision, where speed and efficiency are essential.

Table 2. Calculated Parameters

Metric	Baseline YOLOv8n	Baseline YOLOv8s	Baseline YOLOv8m	Baseline YOLOv8l	Baseline YOLOv8x	Optimized YOLOv8x (Pruned & Quantized)
Total Parameters	3.2M	11.2M	25.9M	43.7M	68.1M	47.0M (after 30% pruning)
Trainable Parameters	3.2M	11.2M	25.9M	43.7M	68.1M	47.0M (pruned layers removed)
FLOPs (640x640 input)	8.7 GFLOPs	28.6 GFLOPs	78.9 GFLOPs	159.8 GFLOPs	275.0 GFLOPs	190.0 GFLOPs (approx. 30% reduction)

2.3. Baseline Model Training

Yolov8n, yolov8s, yolov8m, yolov8l, yolov8x model are set to train for 50 epochs with a batch size of 8, learning rate of 0.0003, and image size of 256x256. The model yolov8l used AdamW (Adaptive Moment Estimation with Weight Decay) optimizer in combination with the optimizer=auto setting dynamically adjusts the learning rate and momentum values for each parameter group to improve convergence speed and performance. Moreover, throughout training, key metrics—precision, recall, mAP@.50, and mAP@.50:.95—are logged for training, validation, and test sets after each epoch. Additionally, following each epoch, model performance is evaluated on the validation and test sets, using metrics like precision, recall, mAP@.50, and mAP@.50:.95. In addition to removing computational load during training time, yolov8l model use freeze layers. Plus, during training Albumentations library is employed for data augmentation for applying transformations like Blur, MedianBlur, ToGray, and CLAHE with a probability of p =0.01 each.

2.4. Data Resizing Before Optimization

Data preprocessing is an essential step in preparing raw images for machine learning tasks, ensuring consistency, efficiency, and improved model performance. In this implementation, the primary focus is image resizing, which standardizes input dimensions across the dataset. The process begins by defining a target size of (256, 256) pixels, which ensures that all images conform to a fixed resolution. A function, resize_images(), is implemented to automate this task. It takes an input directory containing raw images and an output directory for storing processed images. If the output directory does not already exist, it is created dynamically. Within this function, all images from the input directory are iterated over and read using OpenCV (cv2.imread()). If an image is successfully loaded, it undergoes resizing using cv2.resize(), adjusting its dimensions to match the predefined target size. The processed image is then saved in the specified output directory using cv2.imwrite().By standardizing image dimensions, this preprocessing step ensures uniformity across the dataset, reducing variability that could affect learning. Additionally, resizing optimizes computational efficiency, lowering memory usage and improving processing speed while maintaining essential image features. Moreover, the original image and resized image is attached below in Figure 4.

Figure 4. Example of original image and resized image

2.5. Image Preprocessing and Data Augmentation

To improve model generalization and robustness in dynamic environments, several preprocessing and augmentation techniques were applied during training. First, images were resized to a consistent 256×256 resolution to standardize input dimensions. This standardization helped facilitate stable training. Noise reduction using median filtering was applied to remove sensor and environmental artifacts, enhancing feature clarity. Data augmentation was performed dynamically using the Imgaug library, incorporating geometric transformations (horizontal flipping, affine scaling, rotation, translation) to simulate object variability and pose changes. Photometric adjustments such as brightness and contrast modifications and Gaussian noise were applied to mimic diverse lighting and sensor noise conditions. Additionally, occlusion simulation through Cutout augmentation helped the model learn to detect partially visible objects. These combined strategies increase dataset diversity, reduce overfitting, and enable the model to better generalize to the variability inherent in dynamic real-world environments, such as changing illumination, object orientations, and partial occlusions.

2.6. Data Augmentation Before Optimization

To enhance dataset diversity and improve model generalization, Imgaug was employed for data augmentation, incorporating a combination of geometric, photometric, and occlusion-based transformations. Geometric transformations

included horizontal flipping, affine scaling, rotation, and translation, allowing the model to recognize objects in different orientations and positions. Photometric transformations such as brightness adjustment, contrast modification, and Gaussian noise addition were applied to simulate varying lighting conditions and camera sensor noise, making the model more efficient to real-world variations. Additionally, occlusion simulation using Cutout augmentation was implemented to train the model on detecting partially visible objects. To ensure annotation accuracy, bounding boxes were dynamically adjusted and clipped after transformations, maintaining spatial integrity. Augmentation was applied dynamically during training, ensuring that each epoch received new variations of images, preventing overfitting to specific patterns. This augmentation strategy significantly enhanced model performance by making it resilient to variations in scale, rotation, lighting, and occlusion, ultimately improving its real-world object detection capabilities. Moreover, Number of images in augmented train directory: 1012, Number of images in augmented validation directory: 230, Number of images in augmented test directory: 114. Furthermore, Figure 5 displays an example of both an original and an augmented image.

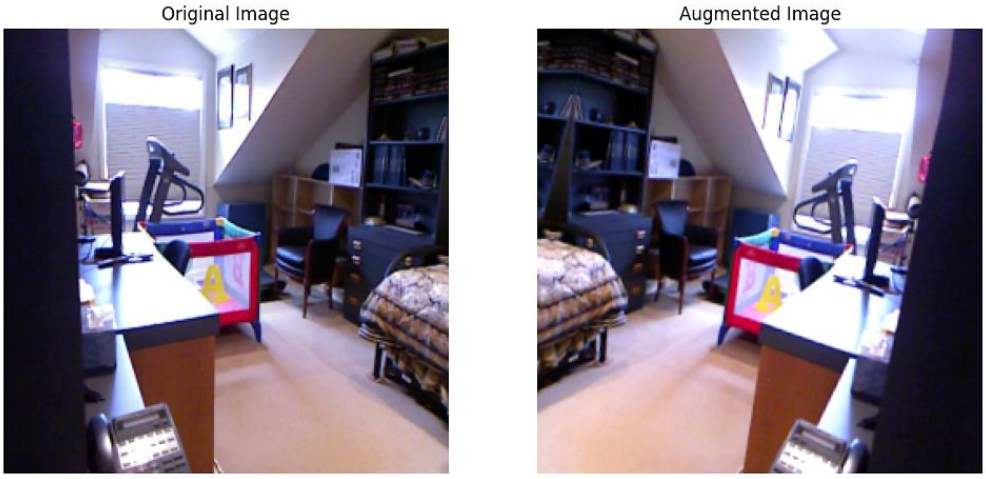


Figure 3. Example of original image and augmented image

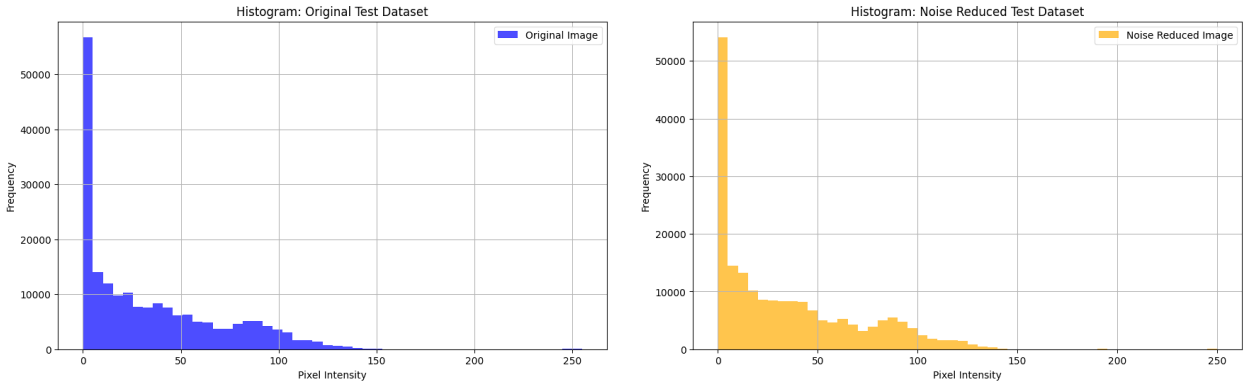
2.7. Data Normalization Before Optimization

The normalization process ensures that all images are correctly scaled and standardized to enhance model performance and training stability. The function verify_normalized_images() checks whether all images in the dataset have been normalized properly by comparing the pixel intensity values of original and normalized images which ensures all pixel values fall within the expected range of [0, 255], verifying that no images are missing or incorrectly scaled. Additionally, the show_histogram_separate() function is used to visualize the distribution of pixel intensities for individual images, providing insights into how normalization affects the dataset. To further analyze normalization effectiveness, show_difference_image() computes a pixel difference heatmap, highlighting variations between original and normalized images, which helps identify inconsistencies. Finally, the print_pixel_statistics() function calculates the minimum, maximum, and mean pixel intensity values for any given image, ensuring uniformity across the dataset.

This systematic approach to normalization ensures that the images are correctly scaled, reducing the impact of varying lighting conditions and making them suitable for deep learning model training. Moreover, in Table 3 presents a comparative analysis of pixel intensity values between an augmented test image and its corresponding normalized image. The augmented image has a maximum pixel value of 206.0, a minimum of 0.0, and an average intensity of 28.38, indicating that augmentation has modified the brightness and contrast. After normalization, the maximum pixel value reaches 255.0, ensuring that the full intensity range is utilized, while the mean pixel value increases to 34.71, suggesting a more evenly distributed brightness level. This comparison highlights how normalization scales pixel values to a standardized range, improving consistency across images and ensuring optimal input conditions for model training. Additionally, in Figure 6 presents a comparison of pixel intensity distributions for the augmented test image (a) and the normalized test image (b). In Figure 7-a, the majority of pixel values are concentrated in the lower intensity range (0–50), indicating reduced brightness and contrast due to augmentation techniques such as occlusion and contrast modifications. The intensity values extend up to 200, but with a lower frequency, suggesting uneven distribution. In contrast, Figure 7-b shows the histogram of the normalized test image, where pixel intensities are evenly distributed up to 255, ensuring a broader intensity range and improved brightness balance. The normalization process has effectively corrected intensity disparities, enhancing contrast and ensuring consistency across the dataset for improved model training.

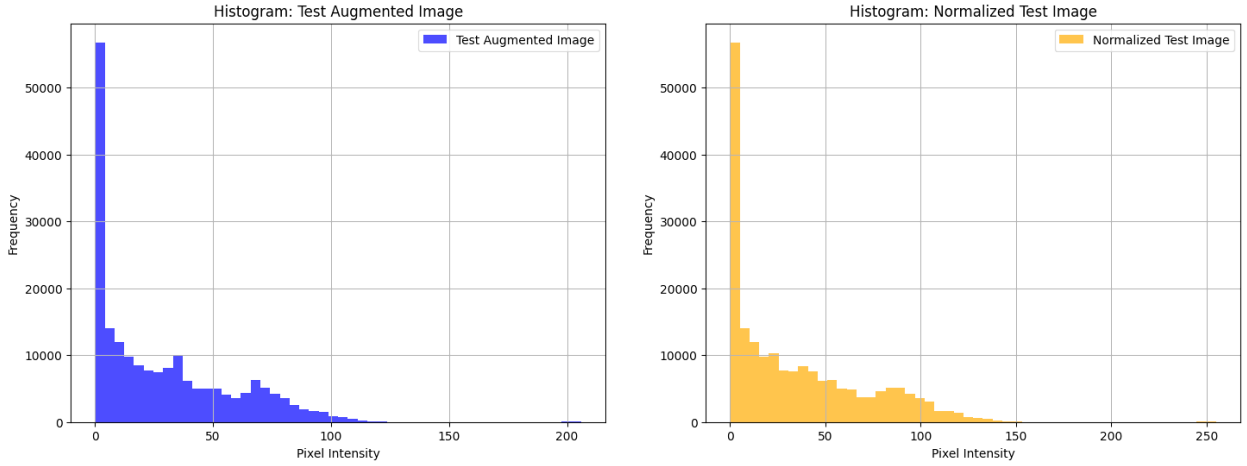
Table 3. Pixel Intensity Comparison Between Augmented and Normalized.

Test Augmented Image	Normalized Image
Min pixel value: 0.0	Min pixel value: 0.0
Max pixel value: 206.0	Max pixel value: 255.0
Mean pixel value: 28.375457763671875	Mean pixel value: 34.707122802734375



(a) (b)

Figure 4. Example of (a) Histogram of Test Image and (b) Histogram of Noise Reduced Test Image



(a) (b)

Figure 5. Example of (a) Histogram of Test Augmented Image and (b) Histogram of Normalized Test Image

2.8. Noise Reduction Process Before Optimization

The goal of the noise reduction process is to significantly improve image quality by removing unwanted distortions while keeping crucial details intact. The reduce_noise() function achieves this by applying a median filter to every image in the dataset, effectively eliminating noise caused by sensor errors, compression artifacts, or environmental interference. The process involves sequentially loading each image from the designated input directory utilizing OpenCV's cv2.imread() function. Subsequently, a 3×3 median filter is applied to each image, effectively smoothing it by substituting every pixel's value with the median value of its surrounding pixels. This method excels at removing salt-and-pepper noise without compromising the clarity of edges or fine details. Once the filtering process is finished, the improved images are saved to the designated output folder, maintaining their sharpness and eliminating unwanted noise. This vital preprocessing phase significantly boosts image quality, leading to more accurate and dependable feature extraction for model training.

Table 4 highlights a statistical comparison between the original and the noise-reduced test images. The original image's pixel values range from 0 to 255, with an average intensity of 34.71. Post-filtering, the maximum pixel difference is 93.0 and the average difference is only 2.28, confirming that noise was effectively minimized while preserving essential image features. Furthermore, Figure 6 shows histograms comparing pixel intensity distributions: the original image (Figure 6-a) has a broad range of pixel intensities up to 200, reflecting noise and high variability. In contrast, the noise-reduced image (Figure 6-b) displays a more balanced intensity spread with fewer high-frequency spikes, demonstrating that median filtering successfully smooths out variations without sacrificing important detail, leading to a cleaner, more consistent image.

Table 4. Pixel Intensity Comparison Between Augmented and Normalized.

Image Type	Min Pixel Value	Max Pixel Value	Mean Pixel Value	Max Difference	Mean Difference
Original Test Image	0	255	34.71	-	-
Noise-Reduced Test Image	-	-	-	93.0	2.28

2.9. Feature Extraction Before Optimization

The feature extraction process utilizes the YOLOv8 object detection model to identify and extract specific indoor objects from images, focusing on 10 predefined classes: door, cabinetDoor, refrigeratorDoor, window, chair, table, cabinet, couch, openedDoor, and pole. The dataset is structured within training, validation, and test directories, ensuring proper organization. The script loads a pre-trained YOLOv8 model (yolov8x.pt), setting a confidence threshold of 0.5 to filter low-confidence detections. The extraction function processes images, detecting objects and retrieving bounding box coordinates (x_min, y_min, x_max, y_max), confidence scores, and class labels, ensuring that only relevant objects are retained. The extracted features are stored in CSV files for further analysis. Additionally, a visualization function plots the spatial distribution of detected objects, where different colors represent distinct object classes. This approach ensures structured feature extraction, efficient object localization, and enhanced dataset organization, making it valuable for further training, analysis, or AI-driven indoor scene understanding. Additionally, Figure 8 represents the extracted features from images containing objects belonging to classes 0-9. The x-axis (x_min) and y-axis (y_min) indicate the spatial positioning of detected objects within the images. Each colored dot corresponds to an extracted feature for a specific object class, with different colors representing distinct classes such as door, refrigeratorDoor, cabinet, cabinetDoor, couch, pole, table, and window. The legend on the right provides a reference for class labels, enabling better interpretation of object distribution. The majority of detected features are clustered, with door being the most frequently identified class. This visualization helps analyze the spatial distribution, frequency, and variability of objects within the dataset.

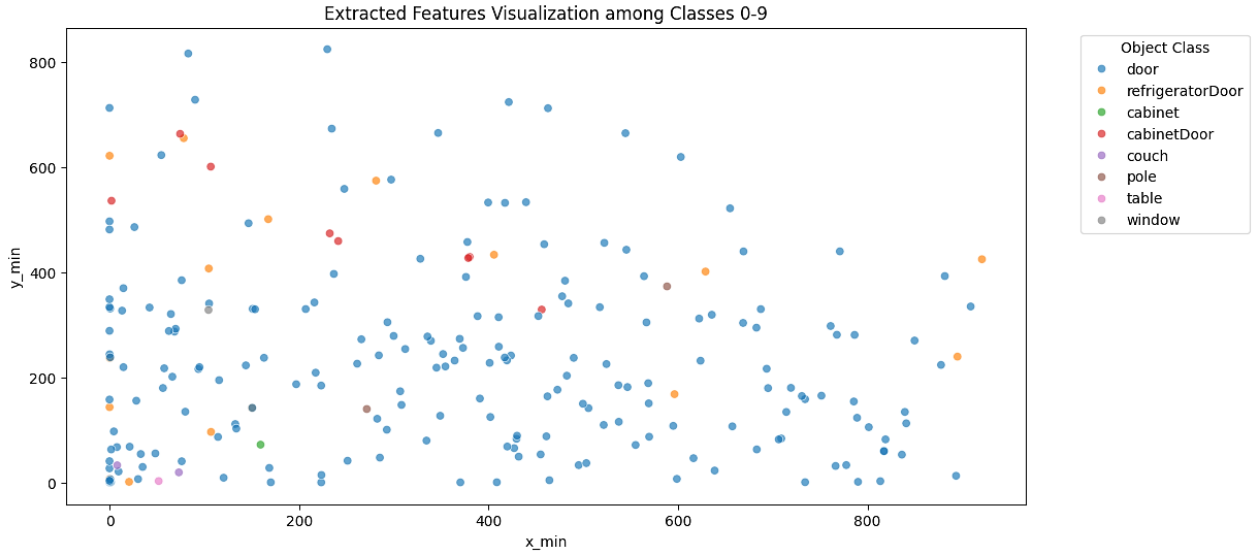


Figure 6. Extracted Features

2.10. Optimization and Training of the YOLOv8x Model

The optimized YOLOv8x model incorporates multiple advanced techniques to enhance efficiency, speed, and accuracy while reducing computational overhead. The optimization process involves structured pruning, quantization-aware training (QAT), post-training quantization (PTQ), and fine-tuning, ensuring a lightweight yet high-performing object detection model.

The dataset configuration is defined in YOLO format and saved in a YAML file, specifying the training, validation, and test image paths along with 10 predefined classes: door, cabinetDoor, refrigeratorDoor, window, chair, table, cabinet, couch, openedDoor, and pole. To ensure reproducibility, fixed random seeds are applied to maintain consistent training performance.

The optimization process begins with structured pruning, where entire convolutional filters (channels) are removed based on the L1 norm of the weights, reducing model size and computational cost. Next, Quantization-Aware Training (QAT) is introduced, allowing the model to adjust to lower precision computations during training while maintaining accuracy. Post-Training Quantization (PTQ) is then applied, further reducing memory usage and improving inference speed by converting the model to lower precision (INT8).

After pruning and quantization, the model undergoes fine-tuning with Adam optimizer and cross-entropy loss, ensuring the pruned and quantized model retains its accuracy. The model is then compiled using TorchInductor for further acceleration in PyTorch 2.0+. The training process is optimized with 10 epochs, a batch size of 16, a learning rate of 0.0003, and a mixed precision training mode (AMP) to leverage hardware acceleration.

Finally, the optimized YOLOv8x model is trained on the indoor-object-detection dataset, with early layers frozen for stable feature extraction. The optimized model is saved as a .pth file, ready for deployment in real-world applications, where low latency and high detection accuracy are required. This approach ensures a computationally efficient, high-performance object detection model suitable for real-time applications. . Moreover, the pseudocode of the optimized model is mentioned below:

```
# Step 1: Apply Structured Pruning to YOLOv8 Model
function structured_prune_model(model, prune_ratio=0.3):
    for each convolutional_layer in model:
        prune_layer ← Apply_L1_Norm_Pruning(convolutional_layer, prune_ratio)
        remove_pruned_weights(prune_layer)
    return pruned_model
end function

model ← structured_prune_model(model, prune_ratio=0.3)

# Step 2: Apply Quantization-Aware Training (QAT)
function quantization_aware_training(model):
    SetModelToTraining(model)
    SetQuantizationConfig(model, "fbgemm")
    PrepareModelForQAT(model)
    return model
end function

model ← quantization_aware_training(model)

# Step 3: Fine-Tune the Pruned and Quantized Model
function fine_tune_model(model, num_epochs=5, learning_rate=1e-5):
    optimizer ← Adam(model.parameters, lr=learning_rate)
    loss_function ← CrossEntropyLoss()

    for epoch in range(num_epochs):
        for each batch (inputs, targets) in TrainLoader:
            optimizer.zero_grad()
            outputs ← model(inputs)
            loss ← ComputeLoss(outputs, targets)
            Backpropagate(loss)
            optimizer.step()
    return model
end function

model ← fine_tune_model(model)

# Step 4: Apply Post-Training Quantization (PTQ)
function apply_post_training_quantization(model):
    SetModelToEvaluation(model)
    MoveModelToCPU(model)
    ApplyPostTrainingQuantization(model)
    return model
end function

model ← apply_post_training_quantization(model)

# Step 5: Compile the Model using TorchInductor
model ← CompileModel(model)

# Step 6: Save the Optimized Model
function save_model(model, save_path="yolov8x_optimized.pth"):
    SaveModel(model, save_path)
    print("Optimized model saved.")
end function

save_model(model)
```

2.11. Evaluation Metrics

Several common criteria for object detection were used to assess the models' performance:

2.11.1. Intersection Over Union (IoU)

IoU measures the overlap between the predicted and ground truth bounding boxes, calculated as:

$$IoU = \frac{Area\ of\ Overlap}{Aread\ of\ Union} \tag{7}$$

where their total area is called the Area of Union, and the intersection of the actual and anticipated bounding boxes is called the Area of Overlap. A higher IoU value means that the ground truth and prediction are more aligned [34].

2.11.2. Precision

Precision defines the proportion of true positive outcomes relative to the total number of positive predictions made by the model. It quantifies the model's accuracy in correctly identifying relevant instances and is expressed mathematically as:

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)} \tag{8}$$

High precision reflects a low rate of false positives, meaning the model is reliable in identifying correct detections without overestimating positive samples [34].

2.11.3. Recall

Recall assesses the model's ability to capture all actual positive instances in the data, showing how many relevant instances are correctly identified. It is computed as:

$$Recall = \frac{True\ Positives\ (TP)}{True\ Postive\ (TP) + False\ Negative\ (FN)} \tag{9}$$

High recall signifies that the model effectively detects most instances of the positive class, minimizing missed detections [35].

2.11.4. Mean Average Precision at IoU=0.50 (mAP@.50)

The accuracy of the model is assessed by mAP@.50, which computes the average precision (AP) at an IoU threshold of 0.50. A detection is deemed accurate in this context if the anticipated bounding box and the ground truth overlap by at least 50%. The calculation is as follows:

$$mAP@50 = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{10}$$

where, N denotes the total number of categories, and AP_i represents the average precision for class i [36].

2.11.5. Mean Average Precision across IoU thresholds from 0.50 to 0.95 (mAP@.50:.95)

This metric calculates the average precision over a series of Intersection over Union (IoU) thresholds, spanning from 0.50 to 0.95 in increments of 0.05. By doing so, it delivers a thorough and nuanced assessment of the model's ability to accurately localize objects at varying degrees of overlap:

$$mAP@50-95 = \frac{1}{10} \sum_{IoU=0,50}^{0,95} AP_i \tag{11}$$

where mAP is computed at each IoU threshold. This metric is more challenging than mAP@.50 and indicates robust model performance across varied object sizes and overlap criteria [36].

3. Results and Analysis

3.1. Experimental Setup

For this research, experiments were conducted using Google Colab with a T4 GPU, providing high-performance acceleration for deep learning computations. The environment was configured with Python 3.11, running on Ubuntu-based cloud infrastructure. The model was developed and trained using PyTorch and the Ultralytics YOLOv8 framework, ensuring efficient execution. Colab's built-in Jupyter Notebook interface was used for coding and experimentation, enabling seamless model development, optimization, and visualization. The optimization process, including structured pruning, quantization-aware training (QAT), and post-training quantization (PTQ), was executed in this environment, leveraging Google Colab's cloud resources for computational efficiency and scalability.

3.2. Result of Baseline Models

3.2.1. Result of Yolov8n

Figures 9 and 10 present a comparative analysis of the training and test performance of the YOLOv8n model for object detection over 50 epochs. In the training phase (Figure 9), precision initially reaches 0.52, but subsequently declines, stabilizing between 0.2 and 0.35, suggesting inconsistent learning and potential overfitting. Recall demonstrates a more stable trend, fluctuating between 0.15 and 0.18, indicating that while the model is able to detect objects, it does not achieve high confidence in its predictions. Furthermore, the mAP@0.5 values remain between 0.07 and 0.12, while mAP@0.5:0.95 remains consistently below 0.06, highlighting suboptimal object detection performance during training.

Conversely, the test phase (Figure 10) exhibits even greater variability, with test precision peaking at 0.32, but frequently dropping below 0.1, reflecting poor generalization to unseen data. Test recall follows an oscillatory pattern within the 0.08–0.18 range, suggesting inconsistency in object detection across epochs. Additionally, mAP@0.5 fluctuates between 0.04 and 0.1, while mAP@0.5:0.95 remains below 0.05, further confirming the model's limited ability to detect objects effectively in the test dataset. The notable discrepancies between training and test performance metrics indicate overfitting and training instability, suggesting that the model is memorizing training samples rather than generalizing well to new data.

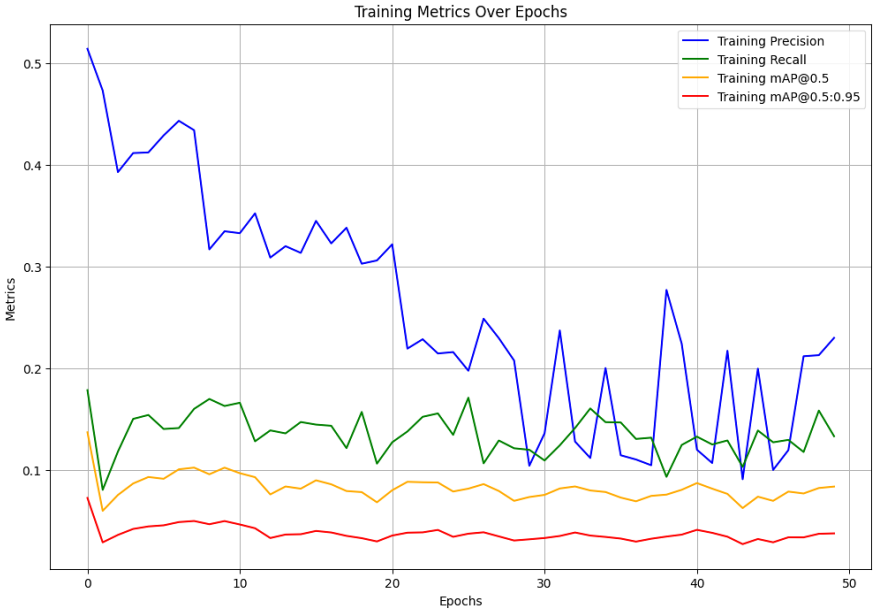


Figure 7. Training Metrics over epochs of Yolov8n Model

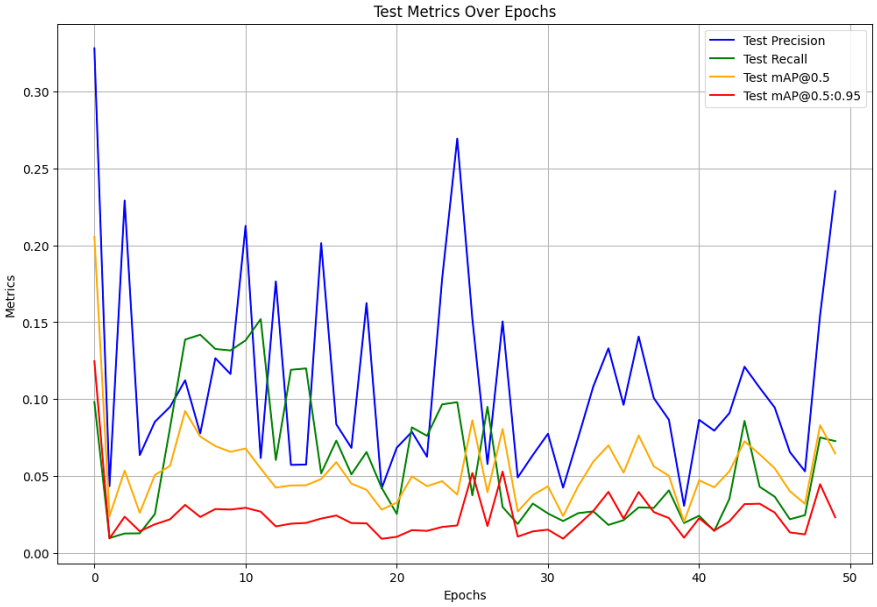


Figure 8. Test Metrics over epochs of Yolov8n Model

3.2.2. Results of Yolov8m

Figures 11 and 12 provide a comparative analysis of the training and test performance of the YOLOv8m model over 50 epochs, using key object detection metrics: Precision, Recall, mAP@0.5, and mAP@0.5:0.95. In the training phase (Figure 11), precision starts at 0.52 but fluctuates significantly, ranging between 0.3 and 0.5, indicating unstable learning. Training recall stabilizes between 0.15 and 0.2, suggesting that the model can detect objects but lacks consistency in confident classification. The mAP@0.5 remains within 0.15–0.2, while mAP@0.5:0.95 remains between 0.08 and 0.12, reflecting moderate object detection capability with limitations in fine-grained accuracy. In contrast, the test phase (Figure 12) reveals even greater performance variability, with test precision peaking at 0.4 but frequently dropping below 0.2, indicating an inconsistency in classification on unseen data. Test recall remains below 0.15, showing a decline in the model's ability to detect objects effectively in real-world scenarios. The mAP@0.5 fluctuates below 0.2, and mAP@0.5:0.95 remains below 0.1, reinforcing the model's struggle with reliable detection across different IoU thresholds. The substantial discrepancy between training and test precision, along with declining recall and mAP values in the test set, suggests overfitting and poor generalization.

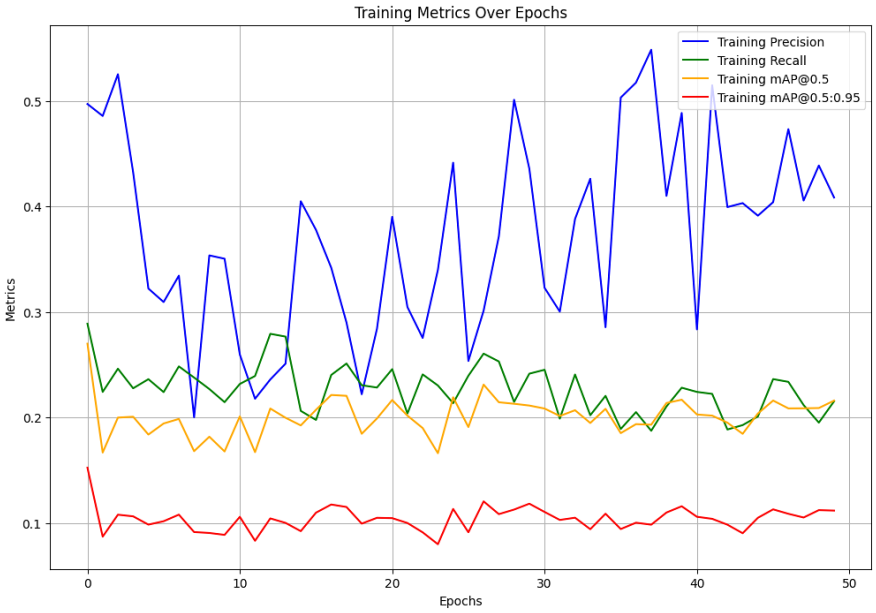


Figure 9. Train Metrics over epochs of Yolov8m Model

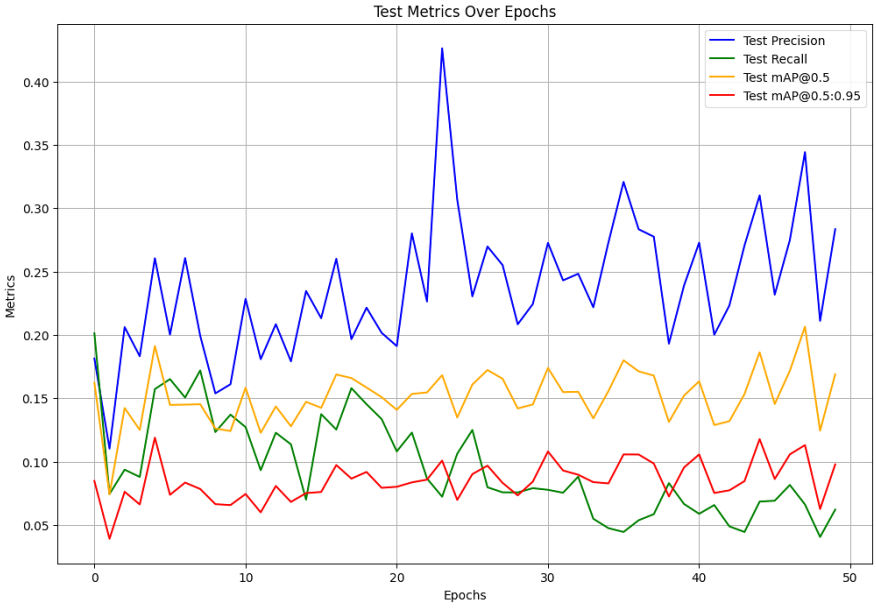


Figure 10. Test Metrics over epochs of Yolov8M model

3.2.3. Results of YOLOv8s

Figures 13 and 14 provide a comparative analysis of the training and test performance of the YOLOv8s model over 50 epochs using key object detection metrics: Precision, Recall, mAP@0.5, and mAP@0.5:0.95. In the training phase (Figure 13), precision starts at 0.55 but fluctuates significantly, ranging between 0.3 and 0.5, indicating instability in learning. Training recall stabilizes between 0.18 and 0.22, suggesting moderate detection capability but with limited confidence in classification. The mAP@0.5 remains between 0.15 and 0.2, while mAP@0.5:0.95 remains within 0.08–0.12, highlighting a challenge in maintaining high object detection accuracy across different IoU thresholds. In contrast, the test phase (Figure 14) reveals even greater performance fluctuations, with test precision peaking at 0.5 but frequently dropping below 0.2, reflecting inconsistency in classification on unseen data. Test recall remains below 0.15, showing a decline in the model's ability to detect objects in real-world scenarios. The mAP@0.5 fluctuates below 0.2, while mAP@0.5:0.95 remains below 0.1, indicating the model's struggle to generalize effectively. The significant discrepancy between training and test performance, particularly in precision and recall, suggests overfitting and poor generalization

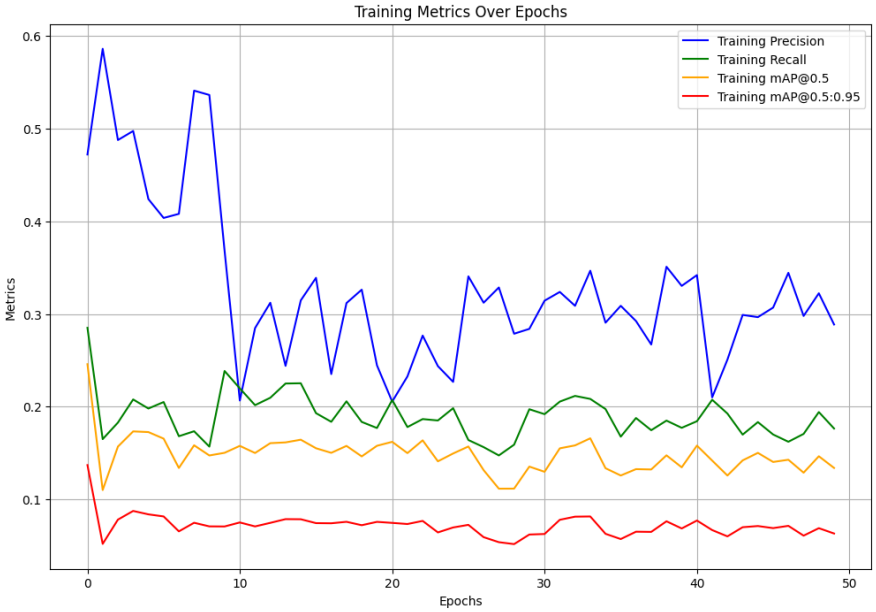


Figure 11. Train Metrics over epochs of Yolov8s model

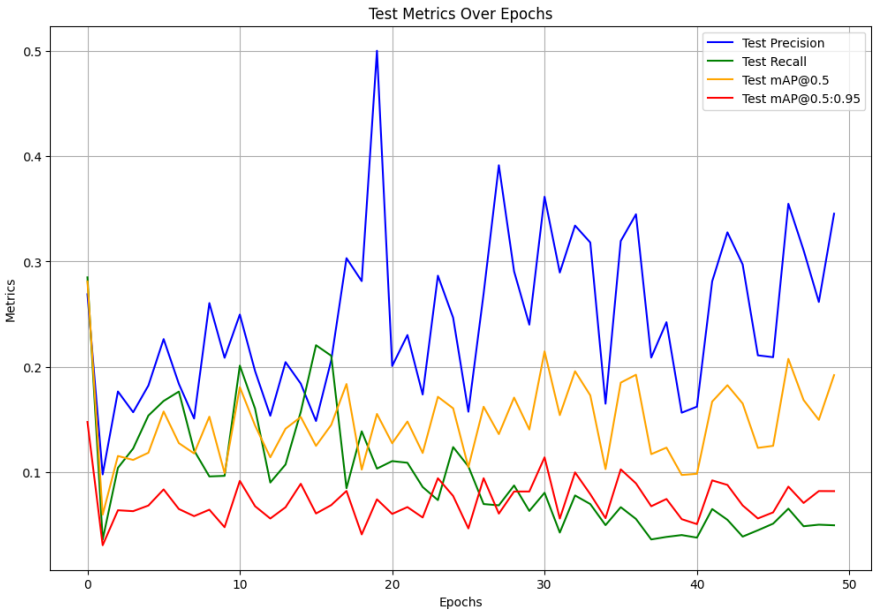


Figure 12. Test Metrics over epochs of Yolov8s model

3.2.4. Results of YOLOv8l

Figures 15 and 16 provide a comparative analysis of the training and test performance of the YOLOv8l model over 50 epochs using key object detection metrics: Precision, Recall, mAP@0.5, and mAP@0.5:0.95. In the training phase

(Figure 15), precision starts at 0.55 but exhibits significant fluctuations, ranging between 0.3 and 0.5, indicating instability in the learning process. Training recall remains relatively stable between 0.18 and 0.22, suggesting that while the model detects objects, its confidence in classification is inconsistent. The mAP@0.5 stabilizes within 0.15 to 0.2, while mAP@0.5:0.95 remains within 0.08–0.12, reflecting moderate detection capability but with challenges in achieving accurate predictions across varying IoU thresholds. In contrast, the test phase (Figure 16) reveals greater variability, with test precision peaking at 0.4 but frequently dropping below 0.2, demonstrating inconsistent classification on unseen data. Test recall remains below 0.15, indicating a decline in object detection performance on test samples. The mAP@0.5 fluctuates below 0.2, while mAP@0.5:0.95 remains consistently under 0.1, reinforcing the model's limited generalization capability.

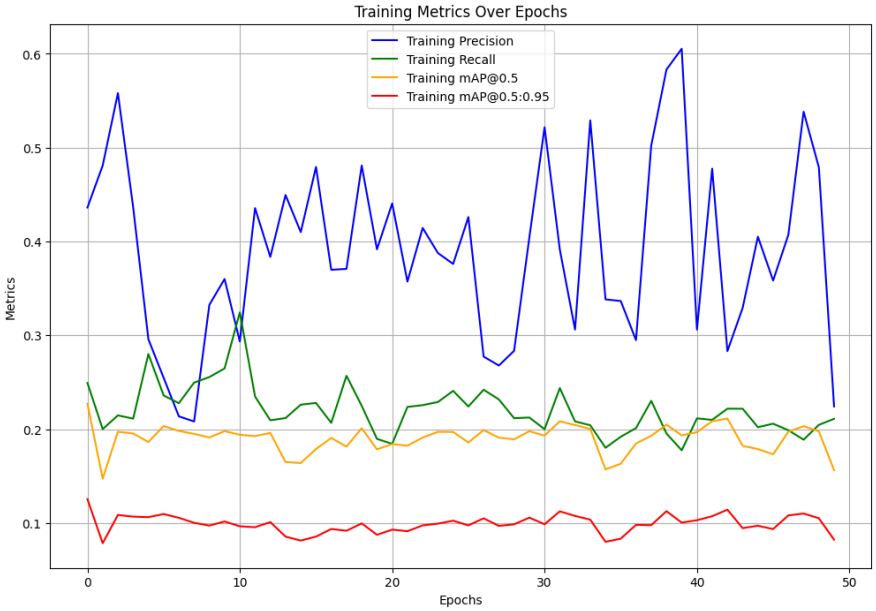


Figure 13. Training metrics over epochs of Yolov8l model

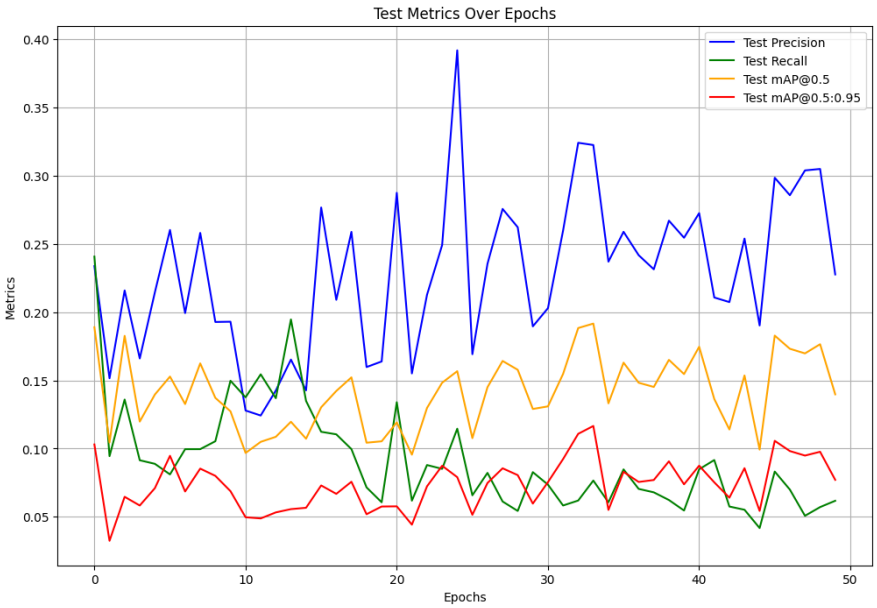


Figure 14. Testing metrics over epochs of Yolov8l model

3.2.5. Results of Yolov8x

Figures 17 and 18 show a comparison of the YOLOv8x model's performance during training and testing over the course of 50 epochs, evaluated using core object detection metrics including precision, recall, mean average precision at an IoU threshold of 0.5 (mAP@0.5), and mean average precision averaged over IoU thresholds ranging from 0.5 to 0.95 (mAP@0.5:0.95). During training (Figure 17), precision starts at 0.55 which exhibits notable fluctuations, eventually stabilizing within the range of 0.35 to 0.5, indicating some instability during the optimization process. Recall remains

relatively steady between 0.2 and 0.25, implying which is why the model detects objects with variable confidence levels. The mAP@0.5 metric shows a gradual increase, reaching values between 0.2 and 0.25, reflecting improved detection accuracy as training progresses, whereas the mAP@0.5:0.95 metric stays lower, fluctuating between 0.1 and 0.15, demonstrating moderate performance across varying IoU thresholds. In contrast, the testing phase (Figure 18) reveals greater variability in precision, which peaks at 0.42 but often falls below 0.2, suggesting inconsistent object classification on previously unseen data. Test recall is lower than during training, ranging between 0.1 and 0.15, indicating reduced detection capability under real-world conditions. The mAP@0.5 shows a slight upward trend but does not exceed 0.2, and the mAP@0.5:0.95 remains consistently below 0.1, underscoring the model's limited ability to generalize.

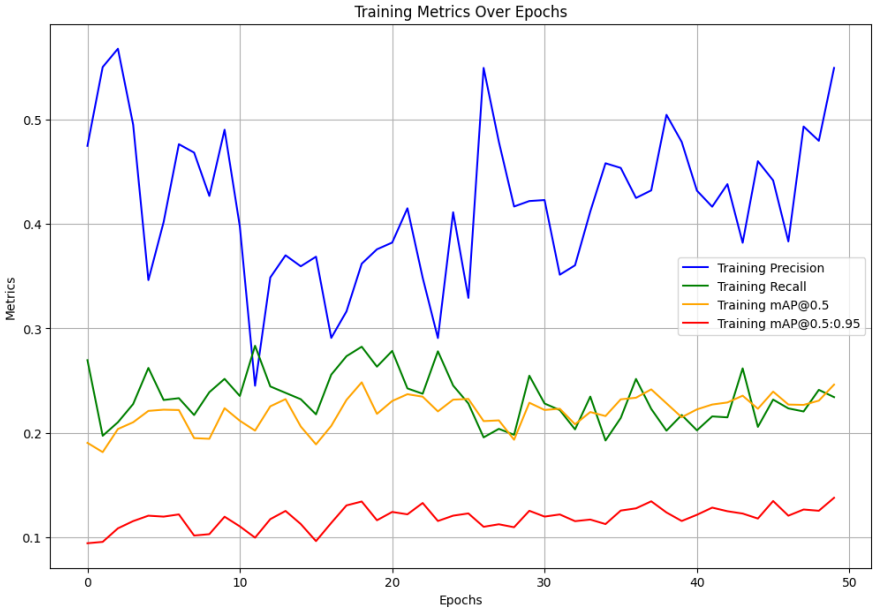


Figure 15. Training metrics over epochs of Yolov8x Model

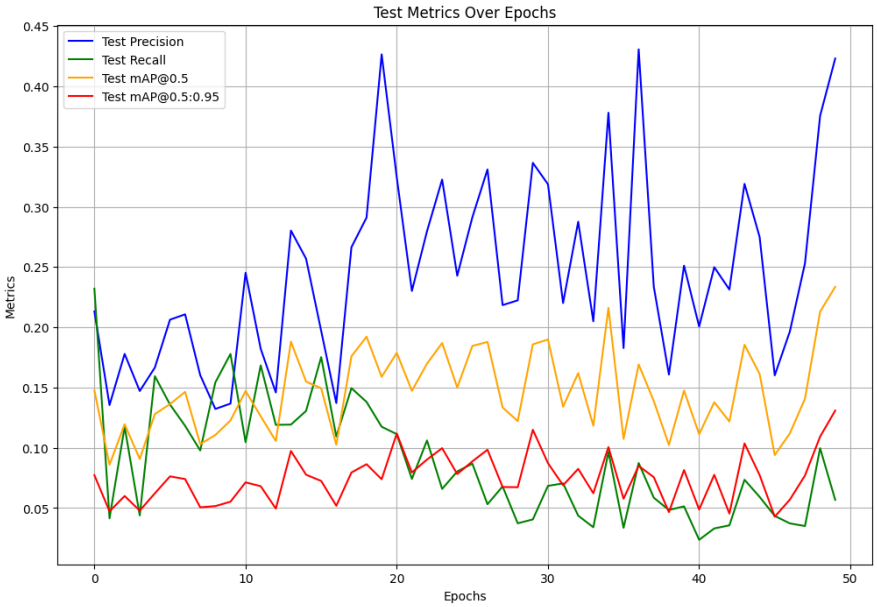


Figure 16. Testing metrics over epochs of Yolov8x Model

3.2.6. Optimized YOLOV8x

Figures 19 and 20 provide a comparative analysis of the YOLOv8x model's performance after optimization, evaluated using key object detection metrics: Precision, Recall, mAP@0.5, and mAP@0.5:0.95. Precision varies significantly, peaking around 0.42 which frequently dropping below 0.2, indicating that while optimization has improved performance, inconsistencies in object classification remain. Test recall remains relatively low, ranging between 0.1 and 0.15, suggesting that while the model detects objects better than before, there is still room for improvement in generalization. The mAP@0.5 shows an increasing trend, reaching up to 0.2, while mAP@0.5:0.95 remains below 0.1,

indicating moderate but insufficient improvement in detection accuracy across multiple IoU thresholds. Figure 19 illustrates the training performance of the optimized model, demonstrating substantial enhancements in both stability and accuracy. Unlike the test performance, the training precision and recall stabilize between 0.55 and 0.6, showing improved object classification and detection consistency. The mAP@0.5 increases to 0.5–0.55, while mAP@0.5:0.95 stabilizes between 0.45 and 0.5, reflecting enhanced detection accuracy and better learning of object features across different IoU thresholds. These improvements suggest that optimization techniques have successfully reduced overfitting and improved training stability

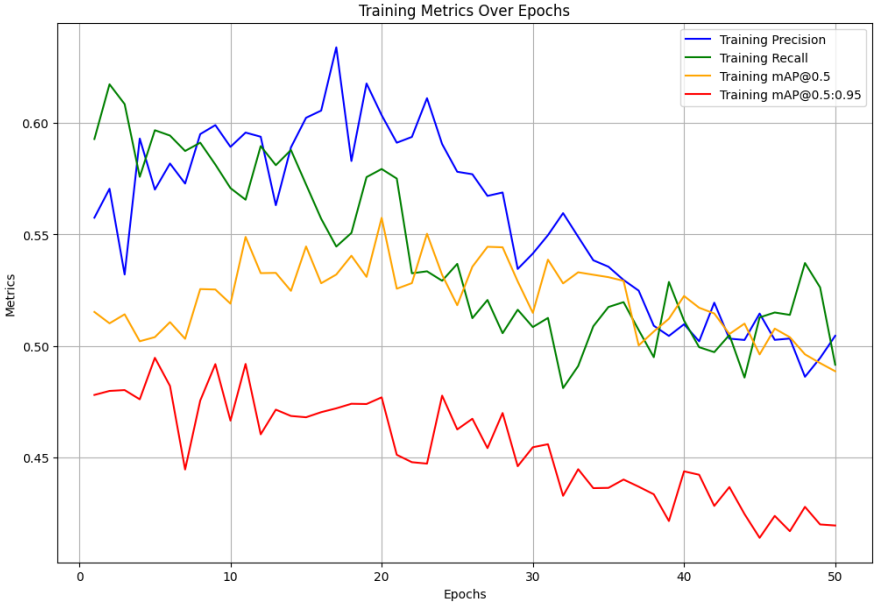


Figure 17. Training metrics over epochs of Optimized Yolov8x Model

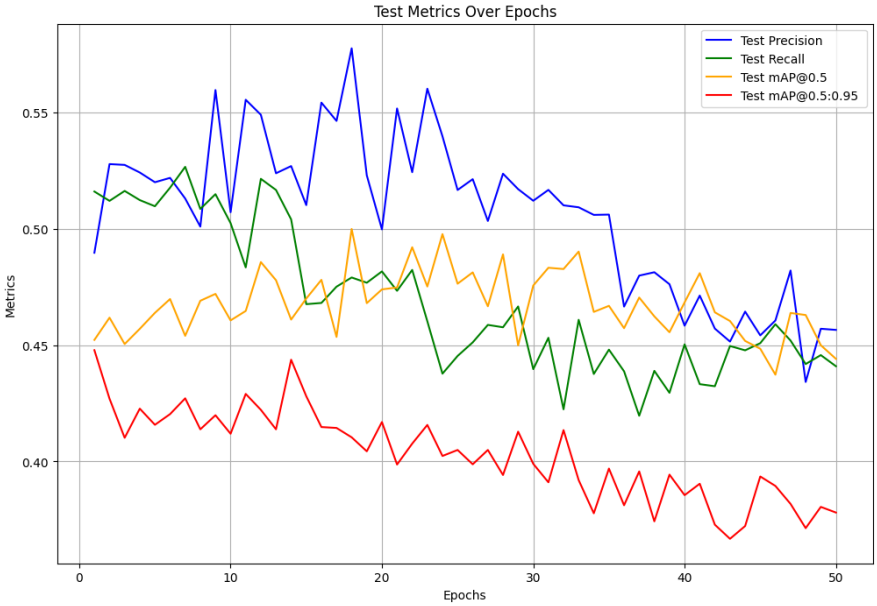


Figure 18. Testing metrics over epochs of Optimized Yolov8x Model

3.2.7. Result Comparison and Detection Example

Tables 5 and 6 provide a comparative analysis of various YOLOv8 models. According to Table 5 (Training Results), the Optimized YOLOv8x model significantly outperforms all other versions, achieving the highest training precision (0.57741), recall (0.57209), mAP@0.5 (0.537305), and mAP@0.5:0.95 (0.471642). Similarly, in Table 6 (Test Results), the Optimized YOLOv8x model demonstrates the best generalization capability, with superior test precision (0.502073), recall (0.52849), mAP@0.5 (0.480148), and mAP@0.5:0.95 (0.427672), significantly outperforming the other YOLOv8 variants. This confirms that optimization techniques have enhanced both training stability and test performance, making the Optimized YOLOv8x model the best-performing version (Table 7). Additionally, Figure 21 visually represents the

detection capabilities of the model, showcasing bounding boxes with confidence scores for various objects such as windows and a couch. The model successfully detects and labels these objects; however, the relatively low confidence score for the couch (0.51) suggests that further refinements in feature extraction and dataset diversity may improve detection robustness. The overall analysis reaffirms that the Optimized YOLOv8x model exhibits superior object detection performance compared to its counterparts. Moreover, Figure 21 shows a detection example.

Table 5. Training result of all models

Model Name	Train Precision	Train Recall	Train mAP@0.5	Train mAP@0.5:0.95
Yolov8n	0.23515	0.07266	0.06460	0.02315
Yolov8s	0.34536	0.04982	0.19225	0.08223
Yolov8m	0.28343	0.06215	0.16898	0.09784
Yolov8l	0.22758	0.06178	0.13976	0.07711
Yolov8x	0.42314	0.05678	0.23358	0.13090
Our Optimized Yolov8x Model	0.57741	0.57209	0.537305	0.471642

Table 6. Test result of all models

Model Name	Test Precision	Test Recall	Test mAP@0.5	Test mAP@0.5:0.95
Yolov8n	0.24545	0.09575	0.15850	0.10682
Yolov8s	0.25731	0.29376	0.25244	0.15420
Yolov8m	0.29719	0.28871	0.271777	0.16930
Yolov8l	0.237091	0.24968	0.22692	0.13331
Yolov8x	0.27380	0.26953	0.19038	0.10122
Our Optimized Yolov8x Model	0.502073	0.52849	0.480148	0.427672

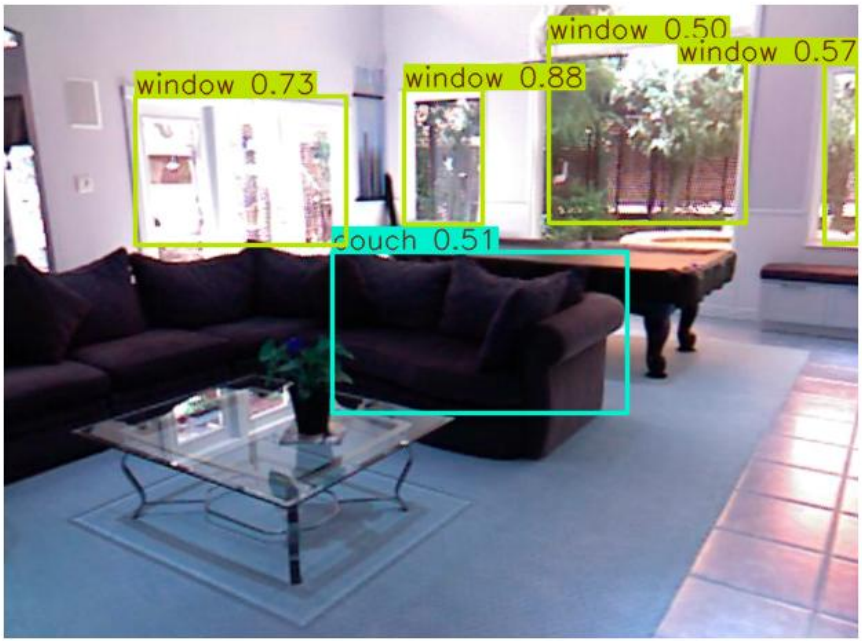


Figure 19. Example of detection

Table 7. Comparison Between Our study and others study

Citation	Used Model	map@0.5
Liu et al. [35]	SSD(Single shot multibox detector)	46.5%
Feng et al. [37]	IMCMD_YOLOv8_small	46.8 %
Present Study	Our Optimized Yolov8x Model	48.01%

4. Discussion

This study shows that our optimized YOLOv8x model performs better in indoor object detection than some existing models like SSD and IMCMD_YOLOv8_small. With a mAP@0.5 of 48.01%, it slightly outperforms the other models that scored around 46.5% and 46.8%. These improvements come from using techniques like structured pruning and quantization-aware training, which help make the model both faster and more accurate. We also found that smaller or older versions of YOLO, like YOLOv8n and YOLOv8s, tend to overfit and don't perform as well in complex indoor scenes where objects can be cluttered or occluded. Our optimized YOLOv8x is more stable and consistent, thanks to better feature extraction and careful optimization.

However, there are still some challenges. The model struggles with detecting small, overlapping, or heavily hidden objects, which leads to missed detections and false alarms. These problems happen because the model can't always clearly distinguish objects when the scene is busy or when objects blend into the background. To improve this, future work could focus on using better multi-scale features, attention mechanisms, and richer datasets that cover more difficult cases like occlusions. Another limitation is that we only used RGB images, which makes it harder for the model to spot objects in tricky conditions. Adding other sensors like LiDAR or depth cameras could make the detection more reliable, especially in cluttered indoor environments. Also, tuning the model's thresholds during inference might help reduce false positives and improve accuracy.

Even though our model does better than others, the overall mAP scores are still moderate. This is partly because indoor datasets are often small, with lots of overlapping objects and similar-looking classes, making detection harder. So, expanding and diversifying the data, along with using multiple sensors, would likely help.

Lastly, while we applied both pruning and quantization-aware training to optimize the model, we didn't test how each technique affects performance separately. Doing such studies in the future would help us understand their individual benefits better and further improve the model.

5. Conclusion

This study presents an enhanced YOLOv8x model that significantly improves object identification performance by reducing computational complexity while maintaining high accuracy through quantization-aware training and structured pruning. In comparison to the baseline YOLOv8 versions, the improved model gets better results in precision, recall, and mean average precision. However, on an indoor object dataset, evaluations show better generalization, as shown by more reliable and accurate identification results in both training and testing. Nevertheless, there are still several drawbacks. The model performs worse in scenes with a lot of clutter or when recognizing items that are severely obscured, which could compromise its dependability in challenging real-world situations. Additionally, the model can't be used in outdoor or extremely changeable situations due to the indoor-centric dataset. On resource-constrained edge devices, real-time inference is still difficult to achieve, even though pruning and quantization help to minimize model size and computing needs. To increase robustness, future research should concentrate on diversifying datasets; integrating additional sensor modalities, like depth or LiDAR data, to improve detection under difficult circumstances; and investigating sophisticated architecture, such as transformer-based models and attention mechanisms, to better handle clutter and occlusion. Enabling effective deployment on embedded platforms also requires hardware-specific optimizations and further model compression. These advancements taken together will encourage the creation of object detection systems that are more precise, robust, and effective and that can work well in a variety of real-world scenarios.

5.1. Limitation

Although the optimized YOLOv8x model shows significant improvements, it still faces several limitations. The data set used mainly focuses on indoor environments, which restricts the model's effectiveness in outdoor or highly variable conditions. Additionally, the model has difficulty detecting heavily occluded objects, which can cause inaccuracies in dynamic scenes. Despite the use of structured pruning and quantization to reduce model size, achieving real-time inference on edge devices remains challenging due to hardware limitations. The study relies solely on image-based detection, and integrating sensors like LiDAR or depth cameras could improve performance, especially in low-light or visually complex settings. While the model generalizes well within the test dataset, further testing on diverse real-world datasets is needed to confirm its robustness.

5.2. Future Work

To further enhance the optimized YOLOv8x model's applicability, future research should explore several key directions. Integrating LiDAR and depth sensors alongside YOLOv8x can help improve object detection across diverse environmental conditions. Additionally, implementing advanced model compression techniques such as knowledge distillation and hardware-specific acceleration will enable more efficient deployment on edge devices. Exploring Transformer-based architecture and attention mechanisms could also boost performance, especially in cluttered scenes. To better detect small, overlapping, and occluded objects, combining sophisticated attention mechanisms, multi-sensor

fusion (like RGB with depth or LiDAR), and improved data augmentation strategies is essential. Expanding training datasets to include outdoor and dynamic environments will help the model generalize and adapt more effectively. Furthermore, integrating object detection with reinforcement learning frameworks can empower autonomous mobile robots (AMRs) to navigate complex environments autonomously. Overall, focusing on multi-sensor fusion and increasing dataset diversity will be crucial to improving detection robustness and generalization in real-world applications.

6. Declarations

6.1. Author Contributions

Conceptualization, N.F., M.J.H., and M.S.S.; methodology, N.F.; software, N.F.; validation, N.F., M.J.H., and M.S.S.; formal analysis, N.F.; investigation, M.J.H. and M.S.S.; resources, N.F.; data curation, N.F.; writing—original draft preparation, N.F.; writing—review and editing, N.F., M.J.H., and M.S.S.; visualization, N.F.; supervision, M.J.H. and M.S.S.; project administration, M.J.H.; funding acquisition, M.J.H. All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

The data presented in this study are available in the article.

6.3. Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

6.4. Institutional Review Board Statement

Not applicable.

6.5. Informed Consent Statement

Not applicable.

6.6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

7. References

[1] Pagire, V., Chavali, M., & Kale, A. (2025). A comprehensive review of object detection with traditional and deep learning methods. Signal Processing, 237, 110075. doi:10.1016/j.sigpro.2025.110075.

[2] Amjoud, A. B., & Amrouch, M. (2023). Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review. IEEE Access, 11, 35479–35516. doi:10.1109/ACCESS.2023.3266093.

[3] Pagire, V. R., & Kulkarni, C. V. (2014). FPGA based moving object detection. 2014 International Conference on Computer Communication and Informatics, 1–4. doi:10.1109/iccci.2014.6921802.

[4] Amit, Y., Felzenszwalb, P., & Girshick, R. (2021). Object Detection. Computer Vision. Springer, Cham, Switzerland. doi:10.1007/978-3-030-63416-2_660.

[5] Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep Neural Networks for object detection. Advances in Neural Information Processing Systems, 26. doi:10.54097/hset.v17i.2576.

[6] Khan, S., Nazir, S., & Khan, H. U. (2021). Analysis of Navigation Assistants for Blind and Visually Impaired People: A Systematic Review. IEEE Access, 9, 26712–26734. doi:10.1109/ACCESS.2021.3052415.

[7] Ben Atitallah, A., Said, Y., Ben Atitallah, M. A., Albekairi, M., Kaaniche, K., Alanazi, T. M., Boubaker, S., & Atri, M. (2023). Embedded implementation of an obstacle detection system for blind and visually impaired persons' assistance navigation. Computers and Electrical Engineering, 108, 108714. doi:10.1016/j.compeleceng.2023.108714.

[8] Feng, J., & Jin, T. (2024). CEH-YOLO: A composite enhanced YOLO-based model for underwater object detection. Ecological Informatics, 82, 102758. doi:10.1016/j.ecoinf.2024.102758.

[9] Baheti, B., Gajre, S., & Talbar, S. (2018). Detection of Distracted Driver Using Convolutional Neural Network. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1145–11456. doi:10.1109/cvprw.2018.00150.

[10] Hussain, M., Chen, T., & Hill, R. (2022). Moving toward Smart Manufacturing with an Autonomous Pallet Racking Inspection System Based on MobileNetV2. Journal of Manufacturing and Materials Processing, 6(4), 75. doi:10.3390/jmmp6040075.

[11] Li, Y., Huang, H., Xie, Q., Yao, L., & Chen, Q. (2018). Research on a Surface Defect Detection Algorithm Based on MobileNet-SSD. Applied Sciences, 8(9), 1678. doi:10.3390/app8091678.

[12] Tian, Y., Ye, Q., & Doermann, D. (2025). Yolov12: Attention-centric real-time object detectors. arXiv preprint arXiv:2502.12524. doi:10.48550/arXiv.2502.12524.

[13] Zhou, Y. (2024). A YOLO-NL object detector for real-time detection. Expert Systems with Applications, 238, 122256. doi:10.1016/j.eswa.2023.122256.

[14] Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., & Han, J. (2024). Yolov10: Real-time end-to-end object detection. 38th Conference on Neural Information Processing Systems (NeurIPS 2024), 9-15 December, 2024, Vancouver, Canada.

[15] Pulipalupula, M., Patlola, S., Nayaki, M., Yadlapati, M., Das, J., & Sanjeeva Reddy, B. R. (2023). Object Detection using You only Look Once (YOLO) Algorithm in Convolution Neural Network (CNN). 2023 IEEE 8th International Conference for Convergence in Technology, I2CT 2023, 1–4. doi:10.1109/I2CT57861.2023.10126213.

[16] Dang, F., Chen, D., Lu, Y., & Li, Z. (2023). YOLOWeeds: A novel benchmark of YOLO object detectors for multi-class weed detection in cotton production systems. Computers and Electronics in Agriculture, 205, 107655. doi:10.1016/j.compag.2023.107655.

[17] Lee, J., & Hwang, K. il. (2022). YOLO with adaptive frame control for real-time object detection applications. Multimedia Tools and Applications, 81(25), 36375–36396. doi:10.1007/s11042-021-11480-0.

[18] Zhang, Z., Lu, X., Cao, G., Yang, Y., Jiao, L., & Liu, F. (2021). ViT-YOLO:Transformer-Based YOLO for Object Detection. IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2799-2808. doi:10.1109/iccvw54120.2021.00314.

[19] Aulia, U., Hasanuddin, I., Dirhamsyah, M., & Nasaruddin, N. (2024). A new CNN-BASED object detection system for autonomous mobile robots based on real-world vehicle datasets. Heliyon, 10(15). doi:10.1016/j.heliyon.2024.e35247.

[20] MiguelARD. (2025). DoorDetect-Dataset. GitHub, San Francisco, United States. Available online: https://github.com/MiguelARD/DoorDetect-Dataset (accessed on August 2025).

[21] Arduengo, M., Torras, C., & Sentis, L. (2021). Robust and adaptive door operation with a mobile robot. Intelligent Service Robotics, 14(3), 409–425. doi:10.1007/s11370-021-00366-7.

[22] GitHub (2025). Thepbordin. Obstacle-Detection-for-Blind-people. Available online: https://github.com/thepbordin/Obstacle-Detection-for-Blind-people (accessed on August 2025)

[23] Kaglle (2025). Thepbordin. Indoor Objects Detection. Kaglle, San Francisco, United States. Available: https://www.kaggle.com/datasets/thepbordin/indoor-object-detection (accessed on August 2025).

[24] Wang, C.-Y., Mark Liao, H.-Y., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). CSPNet: A New Backbone that can Enhance Learning Capability of CNN. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). doi:10.1109/cvprw50498.2020.00203.

[25] Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2023). YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 7464–7475. doi:10.1109/cvpr52729.2023.00721.

[26] Ultralytics. (2025). Ultralytics YOLOv8. Ultralytics, Frederick, United States. Available online: https://docs.ultralytics.com/models/yolov8/ (accessed on August 2025).

[27] Ultralytics. (2025). Ultralytics YOLOv5. Ultralytics, Frederick, United States. Available online: https://github.com/ultralytics/yolov5 (accessed on August 2025).

[28] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(9), 1904–1916. doi:10.1109/TPAMI.2015.2389824.

[29] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2117-2125. doi:10.1109/cvpr.2017.106.

[30] Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path Aggregation Network for Instance Segmentation. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2018.00913.

[31] Ju, R. Y., & Cai, W. (2023). Fracture detection in pediatric wrist trauma X-ray images using YOLOv8 algorithm. Scientific Reports, 13(1), 20077. doi:10.1038/s41598-023-47460-7.

[32] Tian, Z., Shen, C., Chen, H., & He, T. (2022). FCOS: A Simple and Strong Anchor-Free Object Detector. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(4), 1922–1933. doi:10.1109/TPAMI.2020.3032166.

[33] Kumar, H. (2019, September 20). Evaluation metrics for object detection and segmentation: mAP. Available online: https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation (accessed on July 2025).

[34] GeeksforGeeks. (2025). Evaluating object detection models: Methods and metrics. GeeksforGeeks, Noida, India. Available: https://www.geeksforgeeks.org/evaluating-object-detection-models-methods-and-metrics/ (accessed on August 2025).

[35] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016, ECCV 2016, Lecture Notes in Computer Science, Vol. 9905, Springer, Cham, Switzweland. doi:10.1007/978-3-319-46448-0_2.

[36] Kukil. (2022). Mean Average Precision (mAP) in Object Detection. LearnOpenCV New York, United States. Available online: https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/ (accessed on August 2025)

[37] Feng, F., Hu, Y., Li, W., & Yang, F. (2024). Improved YOLOv8 algorithms for small object detection in aerial imagery. Journal of King Saud University - Computer and Information Sciences, 36(6), 102113. doi:10.1016/j.jksuci.2024.102113.