An In-Depth Study of Tiny Object Detection Using Deep Learning Approach

Student Name:	Rahman Md Mahbubur (Saikat)
Student's Number:	3820231103

Subject Name: Deep Learning and Intelligence Image Analysis

Major:	Computer Science and Technology

Abstract

Object detection is a critical computer vision task that involves identifying and localizing objects within images, and it has significant applications in various fields such as autonomous driving and surveillance. Existing research often struggles with achieving high accuracy and low false positive rates simultaneously, especially in diverse and complex datasets.

This study evaluates the performance of various YOLO models, including YOLOv5 and YOLOv8, both in their default and custom configurations, on training and test datasets. The models were assessed using metrics such as average recall, miss rate, false positive rate, precision, and accuracy. The custom YOLOv5s model demonstrated superior performance, achieving the highest average recall (0.8486) and accuracy (0.8475) on the training dataset, with a high precision of 0.9923. On the test dataset, it maintained an average recall of 0.5783 and an average precision of 0.8798, outperforming the other models. Similarly, the custom YOLOv8s model showed excellent results, particularly on the test dataset, with the highest average recall (0.5788) and accuracy (0.5788), a perfect false positive rate (0.0000), and a high precision (0.8372).

These findings indicate that custom configurations of YOLOv5s and YOLOv8s significantly enhance model performance, making them more effective for object detection tasks in diverse applications.

Keyword: Object detection, Deep Learning, Computer Vision, YOLOv5, and YOLOv8

Introduction

The section addresses the significant challenge of moderating small object in the digital age, exacerbated by the global increase in internet usage and the accessibility of surveillance content [1]. It emphasizes the role of computer vision, a blend of artificial intelligence and image processing, in identifying and filtering such content to ensure safer digital spaces [2].

Background

The exponential growth of digital media has introduced significant challenges in the field of surveillance imagery, particularly due to the impracticality of traditional manual control methods. Human moderators face time constraints and limitations, necessitating automated systems to detect small content for maintaining digital safety and regulatory compliance. Computer vision, a subset of artificial intelligence (AI), aims to enable computers to interpret digital images and videos similarly to humans. Within this field, object detection plays a crucial role by recognizing and locating objects in images, vital for applications like social media content moderation (Liu et al., 2020). The development of deep learning, especially Convolutional Neural Networks (CNNs), has advanced object detection models significantly. The YOLO (You Only Look Once) algorithm, introduced by Redmon et al. in 2016, transformed object detection by treating it as a single regression problem, directly converting image pixels into bounding box coordinates and class probabilities.

Recent iterations, such as YOLOv5 and YOLOv8, have demonstrated substantial improvements. Despite being unofficial, YOLOv5 is popular for its speed and accuracy, making it suitable for real-time applications. YOLOv8, the latest iteration, features enhanced architecture and training techniques, improving its ability to detect smaller and more nuanced objects (Zhao & Shen, 2021). Detecting small content remains challenging due to varied scales and occlusions, leading to high false-negative rates. Furthermore, ethical and privacy concerns necessitate robust systems that ensure privacy while providing thorough monitoring (Taylor et al., 2022). YOLOv8's advanced features increase precision and recall rates, making it highly effective for swiftly detecting narrow content, which is crucial for surveillance footage. The evolution of object detection algorithms like YOLOv5 and YOLOv8 promises to enhance digital content moderation, thereby creating safer online environments by enabling more effective and faster moderation.

Why Opt for YOLO in Object Detection Algorithms?

The selection of appropriate models is pivotal to the effectiveness and efficiency of any machine learning-driven research. In this thesis, we explore the use of advanced object detection models, specifically YOLOv5n, YOLOv5s, YOLOv8n, and YOLOv8s for the detection of content within surveillance camera. Selecting YOLOv5 and YOLOv8 was strategic, aiming to capitalize on their advanced capabilities in detecting tiny content efficiently within diverse digital media. These models, which are the latest iterations in the ongoing development of the YOLO series, offer a blend of high accuracy, processing speed, and adaptability that is crucial for our research objectives.

Comparative Analysis and Model Selection

Comparative Analysis: The table below outlines the configuration details for two variants of the YOLO (You Only Look Once) object detection model: YOLOv5 and YOLOv8. These models are renowned for their efficiency in real-time object detection tasks. The table summarizes the key parameters, anchors, backbone architecture, and head design for each model, highlighting their differences and similarities.

Table 1 Architecture of YOLOv5 and YOLOv8

Parameters	Yolov8	Yolov5
Parameters	nc: 6 depth_multiple: 0.33 width_multiple: 0.50	nc: 6 depth_multiple: 0.33 width_multiple: 0.25
Anchors	P3/8: [10,13, 16,30, 33,23] P4/16: [30,61, 62,45, 59,119] P5/32: [116,90, 156,198, 373,326]	--
Backbone	- Focus, Conv, and C3 layers with varying parameters	- Conv and C2f layers with varying parameters
Head	- Conv, nn.Upsample, and Concat layers with varying parameters	- Conv, nn.Upsample, and Concat layers with varying parameters

Model Selection: The decision to utilize YOLOv5 and YOLOv8 was informed by their distinct advantages tailored to specific project needs. YOLOv5 was chosen for its superior accuracy, ideal for scenarios where precision is paramount, such as medical imaging. Conversely, YOLOv8 offers enhanced processing speeds, crucial for real-time applications like surveillance.

Table 2 Comparison: YOLOv5 vs YOLOv8 Models

Model	**Architecture**	**Performance Metrics**	**Adaptability**	**Use Cases**
YOLOv5s	Intermediate size, balances accuracy/speed	Good compromise between precision	Works well across various hardware setups	General-purpose object detection
YOLOv8n	Optimized for efficiency, advanced CSP	More accurate, especially in complex	Ideal for real-time applications	Surveillance, autonomous vehicles, crowd monitoring

YOLOv8s	Similar to YOLOv5s with improvements	Better mAP than YOLOv5s	Suitable for real-time applications	General-purpose detection with improved accuracy

Problem Statement

The rapid expansion of surveillance data has intensified challenges in moderating small-scale content, rendering traditional human methods impractical due to high costs and inefficiency. Automated systems, while beneficial, often suffer from high false-negative rates, particularly in detecting small-scale or obscure content against dynamic backgrounds. This underscores a significant gap in the effectiveness of current content moderation technologies. Advanced object detection algorithms, such as the YOLO series, offer promising solutions by automating content identification and classification. However, even the latest models like YOLOv5 and YOLOv8 face challenges in accurately detecting complex medley content under varied conditions typical in user-generated media, such as fluctuating image sizes, quality, and complex contexts.

Aim and Objectives

This project aims to enhance the detection and moderation of small content in surveillance using advanced object detection algorithms. The primary focus is on how YOLOv8 improves accuracy, speed, and efficiency over YOLOv5 in detecting small, complex background material in digital content. Objectives include quantitatively analyzing performance improvements in terms of precision and recall rates through statistical analysis and comparing the two YOLO versions under similar conditions. It will provide an extensive empirical evaluation of the YOLOv8 model, focusing on its capabilities in detecting tiny content compared to YOLOv5 under various conditions such as image resolution and object size. The YOLOv8 algorithm will be optimized to enhance detection sensitivity for small-scale and partially obscured content, addressing prevalent challenges in dynamic digital environments. Furthermore, the project will establish benchmark standards for AI models in content moderation by documenting detailed comparisons and performance metrics between YOLOv8 and previous models.

Methodology

This section of the paper outlines the methodologies employed in exploring the enhancements and applications of the YOLO (You Only Look Once) object detection algorithms. The primary focus is on evaluating the performance improvements from YOLOv5 through to the latest iteration, YOLOv8, under various conditions and configurations.

Hardware and Software Environment

Hardware Specifications

The table 4-1 outlines the hardware and operating system specifications of the system in use. It includes details about the processor, graphics processing unit (GPU), installed memory (RAM), and supported operating systems. Some experiments also have been run on the google colab because of my GPU availability.

Table 3 Hardware Environment

Component	**Specification**
System	HP
GPU	GeForce RTX 2080 Ti, 11019MiB; 16 CPUs, 125.6 GB RAM, 934.6/937.3 GB disk
CPU	Intel® Core™ i7-10900X CPU @ 3.20GHz × 4 ; 32 GB RAM
Operating System (OS)	Windows 10, Ubuntu 18.04.5 LTS

Software Specifications

This table 2 summarizes the software tools and technologies utilized to support the efficient development and training of machine learning models. Each tool or technology is selected based on its ability to facilitate various aspects of the development process, from programming and code management to enhancing computational performance.

Table 4 Overview of Software Tools and Technologies

Category	**Programming Language**	**GPU Acceleration**	**IDE**	**Version Control**	**Connectivity**	**Image Processing**	**Image Visualization**	**Model Development**
Technology	Python	CUDA and cuDNN	VScode	Git	SSH	OpenCV	Matplotlib	PyTorch

Dataset Description

Training Dataset

The dataset includes 477 images, each annotated to demarcate material with bounding boxes, ensuring detailed and accurate training input. This dataset was collected from the surveillance camera to include various backgrounds such as beach, streets, mining area etc. It is considered challenging due to variations in many factors including human position, instance limitation, and complex backgrounds. The analyzed dataset's correlogram is depicted in Figure 2 (Right), providing a clear visual representation of the correlations between different labels. This correlogram serves as an important analytical tool, offering insights into how various labels within the dataset relate to each other. Each image in the dataset is pre-labeled with bounding boxes that accurately. The dataset two classes person and car have appeared separately in Figure 2 (left). The dataset is imbalanced and it has huge person class instances than the car class.

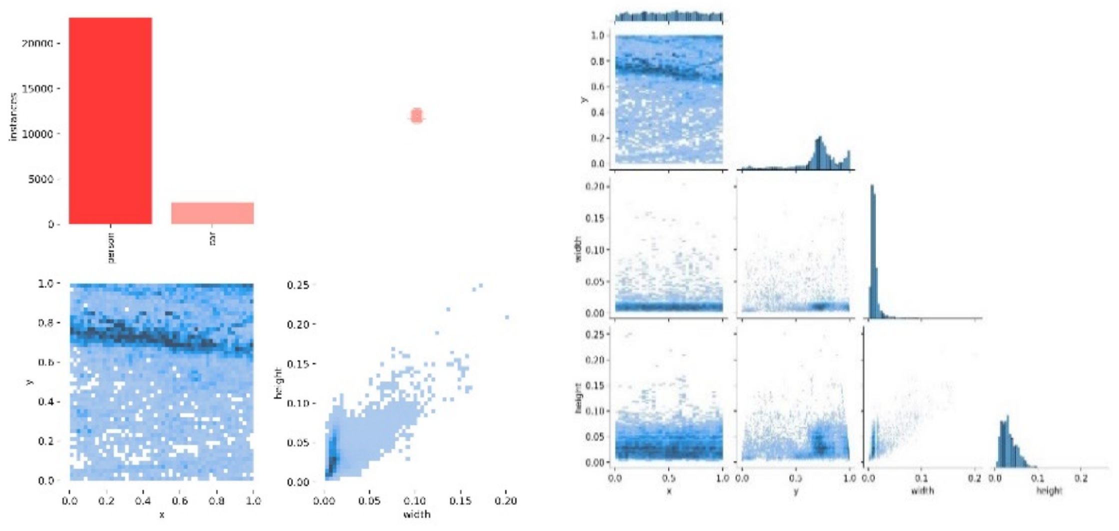


Figure 1 Data set details (left) & dataset's correlogram (right)

Testing Dataset

To evaluate the effectiveness of object detection models in real-world scenarios, a diverse collection of images are gathered from various sources for testing purposes. Unlike the training dataset, these images are also provided with the ground labels for the testing purposes. They are deliberately excluded from the training process and reserved solely for testing the model's ability to generalize and detect the content in unlabelled. This approach ensures that the performance and reliability of the models are assessed based on their capability to identify content without having been exposed to these specific images and videos during the training phase.

Decompose Training Data and Cleaning:

In order to impartially evaluate the performance of neural network model, a dataset containing the same type of object as the trained object is essential. The dataset has been split into two parts – train dataset, valid dataset. The train dataset consisted of 85% images (405) while valid dataset consisted of 15% images (76) from the original dataset. Label text files have the same name as the images. This allows the models to be tested against unseen data, providing a measure of their generalization capabilities.

Ensuring all data are correctly formatted and stored in the YOLOv5 and YOLOv8 folder simplifies access and subsequent cleanup post-training. Prior to training, the dataset undergoes verification to check for corrupted files or improper annotations. This ensures the quality and reliability of the data being fed into the models. We ensure the dataset contains no duplicates to prevent bias in model training.

Configuration for Model Training

Creating Python Virtual Environment and YOLO Repository Copy

We kept our libraries and packages into a new python virtual environment. We activate our environment for installing further dependencies. The YOLOv5 architecture, like its previous versions, was created conceptually and may be accessed through a GitHub repository; we copied repo into our system. YOLOv5 was built by Ultralytics using the PyTorch framework, which is widely respected in the AI world. Therefore, we installed

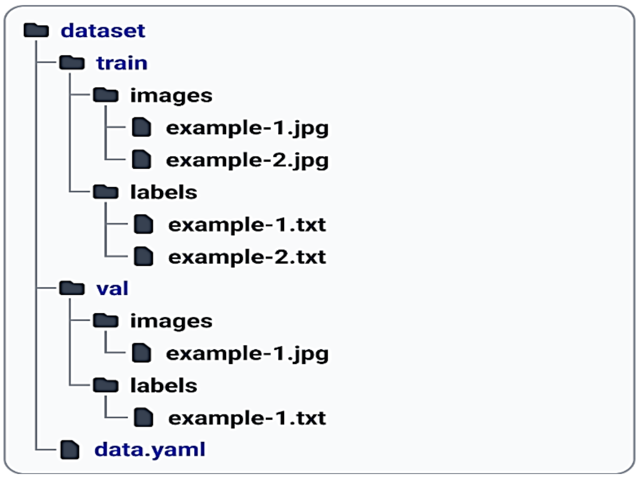
Pytorch and OpenCV into our virtual environment and setup our GPU. Furthermore, we installed all dependencies required for running yolo models by requirements.txt.

Initial Testing and Dataset Folder Structure

Conducting initial tests to verify the correct setup and functionality of the model. This includes running the detect.py script to perform inference tests using pre-trained models, ensuring the model operates as expected. Yolo algorithm have their own specific folder structure and format for feeding algorithm. Snice our data have already normalized into yolo format so we do not have much work onto this.

Configuration of the data.yaml

To facilitate the use of images as input for the YOLO model developed on the PyTorch framework, a `data.yaml` file is essential. In Figure 14 contains summary information about the dataset and instructs the model on where to find the training and validation images. The structure of the `data.yaml` file includes entries for the paths to the



```
dataset
├─ train
│  ├─ images
│  │  ├─ example-1.jpg
│  │  └─ example-2.jpg
│  └─ labels
│     ├─ example-1.txt
│     └─ example-2.txt
├─ val
│  ├─ images
│  │  └─ example-1.jpg
│  └─ labels
│     └─ example-1.txt
└─ data.yaml
```

Figure 2 an Example of dataset structure for yolo

training and validation set directories (`train` and `val`), the number of classes (`nc`), and the names of these classes (`names`). Since our collected dataset does not inherently include a `data.yaml` file, it is necessary to create one manually. Once created, the file is uploaded to the appropriate directory within the YOLO framework to ensure proper linkage and access during model training.

Data Augmentation

Data augmentation techniques applied, such as flipping, rotation, scaling, and color adjustments, to increase the diversity of the training set. We have applied these techniques into both models YOLOv5 and YOLOv8.

Training YOLOv5 and YOLOv8 Models

Yolo have arguments parser functions as a preliminary framework, allowing researchers the flexibility to customize it according to their own requirements. Possible modifications include adding or removing layers, fine-tuning blocks, integrating novel image processing techniques, and adjusting optimization methods or activation functions to improve performance.

First Round Training (Yolov5s)

I trained YOLOv5s, a lightweight object detection model, using its default settings. The process involved preparing a dataset with annotated images, splitting it into training and validation sets, and utilizing YOLOv5s's built-in training script. The default settings optimized the model's hyperparameters, leveraging pre-trained weights for faster convergence. During training, the model learned to detect objects within the provided images, improving its performance over epochs by the loss function. Post-training, the model was evaluated on the validation set, achieving satisfactory results. This experiment demonstrated YOLOv5s's efficiency and ease of use in developing robust object detection systems with minimal configuration. We understood the model's performance by observing its hyperparameters during the initial training. Based on these observations, we then further set up the model's architecture and parameters, adjusting them for a second round of training. This approach allowed us to fine-tune the model, aiming to enhance its detection accuracy and overall performance in subsequent training iterations. We conducted a total of two experiments using default settings for each model, comparing those hyperparameters with our tuned parameters for further refinement.

Second Round Training (Yolov5s_custom)

1. **Hyperparameters and Settings:** Initially, we ran 30 epochs with 100 iterations each to tune the hyperparameters for our custom models. However, after 10 epochs, we halted the

hyperparameter tuning due to the time-consuming nature of the process. Within those 10 epochs, we identified promising hyperparameters. We then combined these insights with previously trained parameters for our experiment. These parameters included image size, batch size, number of epochs, weights, worker configurations, and optimizer choices. Understanding these distinctions is crucial for practitioners aiming to select the most suitable model for their specific object detection tasks. We can fit hyper parameter if we wait longer time for our models best fitness. Since it is our learning purpose, so we just kept 10 iterations.

2. Adjustment and Optimization: The following table presents an adjustment overview of key parameters between YOLOv5 models, four prominent object detection architectures. We have defined confidence score 0.20 and early stopping 100 (patience) for each model. We have added dropout layer in both models.

Yolov5 hyper parameter tuning for each model

Feature	Exp.: 1 (default)	Exp.: 2 (custom)
Model	YOLOv8s (pretrained)	YOLOv8s (pretrained)
Epochs	300	300
Image Size (imgsz)	640	640
Batch Size (batch)	16	32
Optimizer	SGD	AdamW
Hyper-parameters	Default	lr0: 0.00258; lrf: 0.17; momentum: 0.779; weight_decay: 0.00058; warmup_epochs: 1.33; warmup_momentum: 0.86; warmup_bias_lr: 0.0711; box: 0.0539; cls: 0.299; cls_pw: 0.825; obj: 0.632; obj_pw: 1.0; iou_t: 0.2; anchor_t: 3.44; anchors: 3.2; fl_gamma: 0.0; hsv_h: 0.0188; hsv_s: 0.704; hsv_v: 0.36; degrees: 0.0; translate: 0.0902; scale: 0.491; shear: 0.0; perspective: 0.0; flipud: 0.0; fliplr: 0.5; mosaic: 1.0; mixup: 0.0; copy_paste: 0.0

First Round Training (Yolov8)

I trained YOLOv8n and YOLOv8s models, both with default settings and custom adjustments. For YOLOv8n with default settings, the model was trained using the standard hyperparameters, demonstrating robust performance out-of-the-box. I then trained YOLOv8n with custom adjustments, fine-tuning parameters like learning rate, batch size, and epochs, which led to improved detection accuracy. Similarly, YOLOv8s was trained using its default settings, showing good initial results. The custom-trained YOLOv8s involved adjusting parameters based on initial observations, enhancing the model's precision and recall. These experiments highlight the flexibility of YOLOv8 in adapting to specific object detection tasks with both standard and customized configurations.

Second Round Training (Yolov8)

1. **Hyperparameters Settings and Adjustment:** The following table presents a comparative overview of key parameters between YOLOv8 models, four prominent object detection architectures. We have defined confidence score 0.20 and early stopping 100 (patience) for each model. We have generalized hyper parameters upto 30 epoch with 300 iterations. It is time consuming that is why we interrupting within 10 epochs so that we can have a nearest hyper parameters. We can fit hyper parameter if we wait longer time for our models best fitness. Since it is our learning purpose, so we just kept 10 iterations.

Yolov8 hyper parameter tuning for each model

Feature	Exp.: 1 (default)	Exp.: 2 (custom)	Exp.: 3 (default)	Exp.: 4 (custom)
Model	YOLOv8n (pretrained)	YOLOv8n (pretrained)	YOLOv8s (pretrained)	YOLOv8s (pretrained)
Epochs	300	300	300	300
Image Size (imgsz)	640	640	640	640
Batch Size (batch)	16	32	16	32
Freeze	--	[0] (freeze first layer)	--	[0] (freeze first layer)
Dropout	--	0.1	--	0.1
Optimizer	AdamW	SGD	AdamW	SGD
Hyper-parameters	Default	lr0=0.00938, lrf=0.00875, momentum=0.93099, weight_decay=0.00057, warmup_epochs=3.0, warmup_momentum=0.75579, box=7.02336, cls=0.44188, dfl=1.40956, hsv_h=0.0154,	Default	lr0=0.00933, lrf=0.0102, momentum=0.938, weight_decay=0.0005, warmup_epochs=3.05, warmup_momentum=0.85, box=7.6, cls=0.55, dfl=1.5, hsv_h=0.017, hsv_s=0.69,

	hsv_s=0.73327, hsv_v=0.36094, degrees=0.0, translate=0.09004, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.48808, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0	hsv_v=0.45, degrees=0.0, translate=0.10, pose=12.5, kobj=1.1, scale=0.55, shear=0.0,perspective=0.0, flipud=0.0, fliplr=0.55, bgr=0.0, mosaic=1.0, mixup=0.0,copy_paste=0.0

Model Evaluation

The YOLO algorithm's classification performance evaluates using two key metrics: accuracy and F1 score.

1. **Accuracy:** Accuracy is defined as the number of correct predictions over the total number of predictions, particularly effective when target classes are balanced.
2. **F1 Score:** F1 Score is the harmonic mean of precision and recall, useful for assessing accuracy when false positives and false negatives are important. Moreover, Precision is the ratio of true positives to the total predicted positives. Recall is the ratio of true positives to the sum of true positives and false negatives.

Experiment Results & Performance Analysis

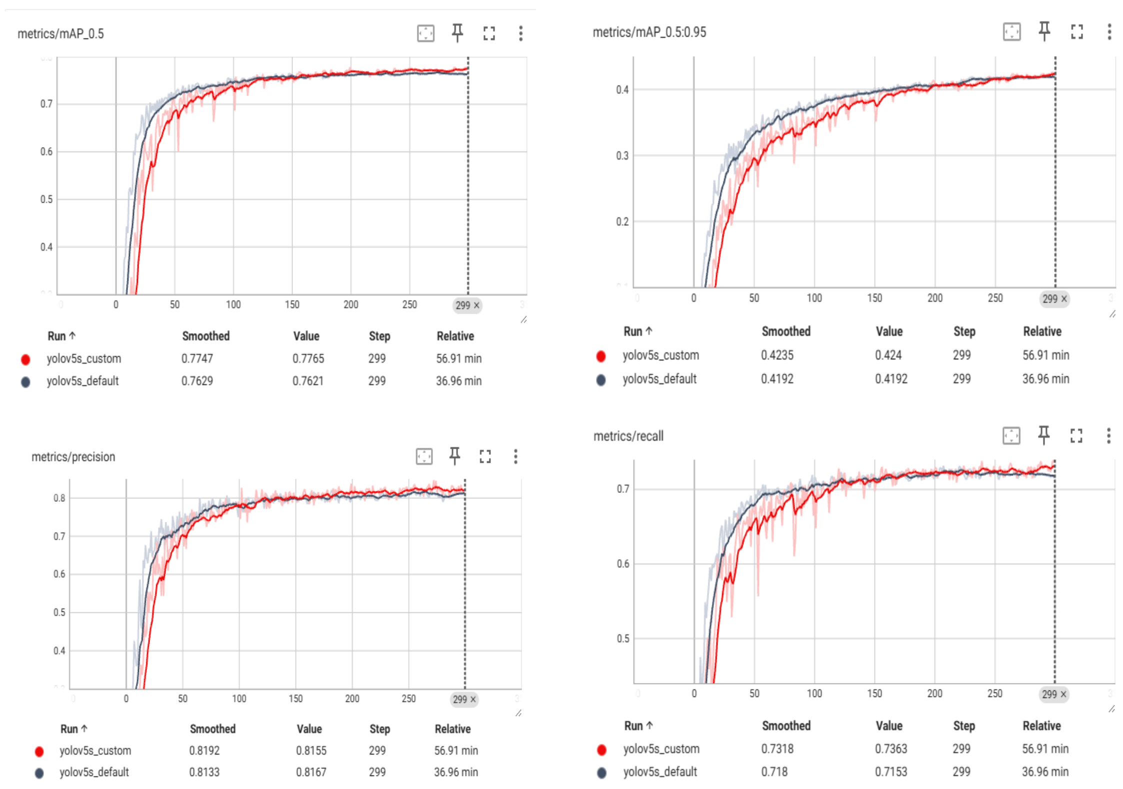
This section details the outcomes of experiments conducted using the YOLOv5 and YOLOv8 frameworks, providing a comprehensive analysis of general findings, and class comparisons.

Training Performance of YOLOv5

The results obtained by YOLOv5 after finished the training process. One to 300 epochs are the representation of the results obtained by the algorithms after evaluating.

Training Metrics Comparison

The below graphs illustrate a comparative analysis of training metrics between YOLOv5 models with default settings and custom configurations. The metrics evaluated include mAP@0.5, mAP@0.5:0.95, precision, and recall. The custom model (represented by the red line) generally shows improved performance over the default model (represented by the blue line) across all metrics. Specifically, the mAP@0.5 and precision metrics show significant improvement with the custom model, indicating enhanced accuracy and detection capability. Both models converge after around 200 epochs, but the custom model consistently outperforms the default model. This comparison underscores the benefits of fine-tuning hyperparameters to achieve better performance in object detection tasks.



metrics/mAP_0.5

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.7747	0.7765	299	56.91 min
● yolov5s_default	0.7629	0.7621	299	36.96 min

metrics/mAP_0.5:0.95

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.4235	0.424	299	56.91 min
● yolov5s_default	0.4192	0.4192	299	36.96 min

metrics/precision

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.8192	0.8155	299	56.91 min
● yolov5s_default	0.8133	0.8167	299	36.96 min

metrics/recall

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.7318	0.7363	299	56.91 min
● yolov5s_default	0.718	0.7153	299	36.96 min

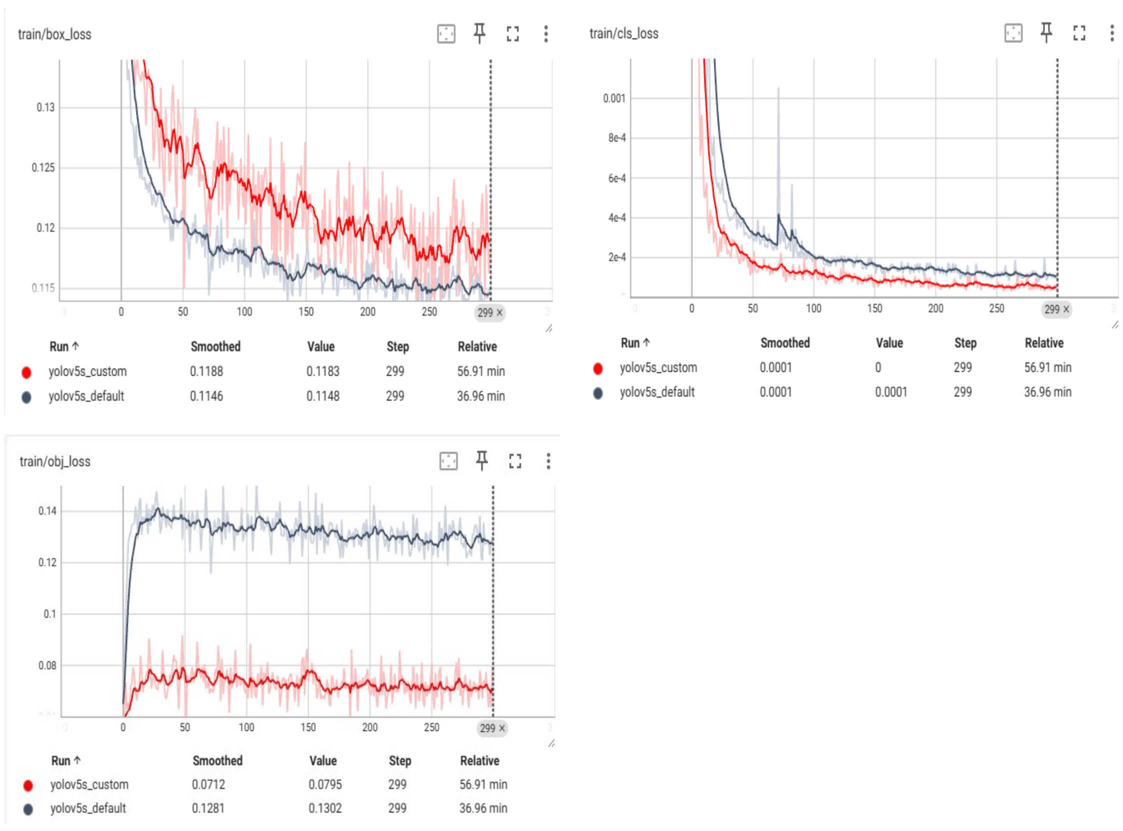
Figure 3 Training Matrices Comparison between default and custom Models

Training Loss Comparison

The graphs in Figure 4 display the training loss of two models, YOLOv5_custom and YOLOv5_default, across three metrics: box loss, class loss, and objectness loss.

For the train/box_loss graph, the YOLOv5_custom model (red) shows a consistently lower loss than the YOLOv5_default model (blue), indicating better performance in predicting bounding boxes. In the train/cls_loss graph, both models exhibit rapid initial decline, with the YOLOv5_custom model stabilizing at a lower loss level compared to the YOLOv5_default model, suggesting improved classification accuracy.

The train/obj_loss graph reveals a more distinct difference; the YOLOv5_custom model achieves significantly lower and more stable objectness loss, demonstrating superior object detection capabilities. Overall, the YOLOv5_custom model outperforms the YOLOv5_default model in all three metrics, as evidenced by the consistently lower loss values across training iterations.



train/box_loss

	Run ↑	Smoothed	Value	Step	Relative
●	yolov5s_custom	0.1188	0.1183	299	56.91 min
●	yolov5s_default	0.1146	0.1148	299	36.96 min

train/cls_loss

	Run ↑	Smoothed	Value	Step	Relative
●	yolov5s_custom	0.0001	0	299	56.91 min
●	yolov5s_default	0.0001	0.0001	299	36.96 min

train/obj_loss

	Run ↑	Smoothed	Value	Step	Relative
●	yolov5s_custom	0.0712	0.0795	299	56.91 min
●	yolov5s_default	0.1281	0.1302	299	36.96 min

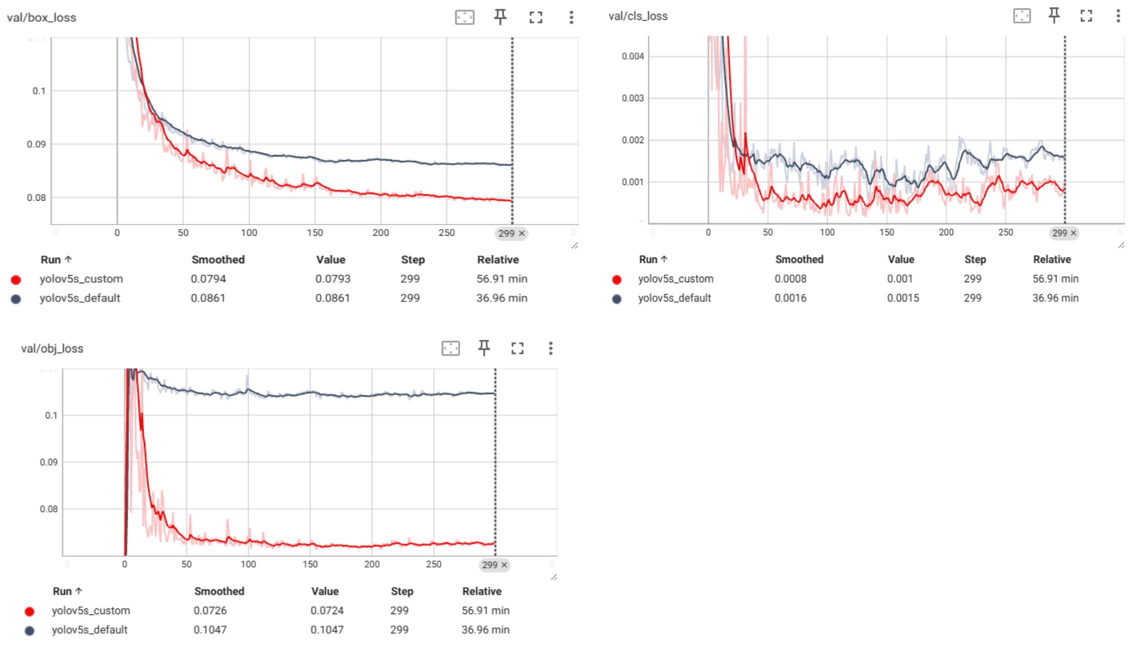
Figure 4 Training Loss between two models

Validation Loss Comparison

The graphs in Figure 5 present the validation loss of two models, YOLOv5_custom and YOLOv5_default, across three metrics: box loss, class loss, and objectness loss.

In the val/box_loss graph, the YOLOv5_custom model (red) demonstrates a lower and more rapidly decreasing loss compared to the YOLOv5_default model (blue), indicating better generalization in predicting bounding boxes on validation data. The val/cls_loss graph shows that the YOLOv5_custom model achieves a consistently lower loss after initial fluctuations, suggesting improved classification accuracy over the YOLOv5_default model, which has higher and more variable loss values.

The val/obj_loss graph highlights a significant difference; the YOLOv5_custom model achieves a notably lower and more stable objectness loss throughout the validation process compared to the YOLOv5_default model. Overall, the YOLOv5_custom model outperforms the YOLOv5_default model in all three metrics, reflecting its superior performance and generalization ability on validation data.



val/box_loss

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.0794	0.0793	299	56.91 min
● yolov5s_default	0.0861	0.0861	299	36.96 min

val/cls_loss

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.0008	0.001	299	56.91 min
● yolov5s_default	0.0016	0.0015	299	36.96 min

val/obj_loss

Run ↑	Smoothed	Value	Step	Relative
● yolov5s_custom	0.0726	0.0724	299	56.91 min
● yolov5s_default	0.1047	0.1047	299	36.96 min

Figure 5 Validation Loss between two models

F1 Confidence of Each Model

The F1-Confidence Curves compare the performance of YOLOv5_default and YOLOv5_custom models across two classes: person and car. For the YOLOv5_default model (left), the F1 score peaks at 0.75 at a confidence of 0.396, with the car class achieving higher F1 values than the person class. In contrast, the YOLOv5_custom model (right) shows a slightly improved peak F1 score of 0.76 at a confidence of 0.385, again with the car class outperforming the person class. Overall, the YOLOv5_custom model demonstrates a marginally better F1 score across all classes, indicating a slight improvement in prediction accuracy over the YOLOv5_default model if we run more epochs.

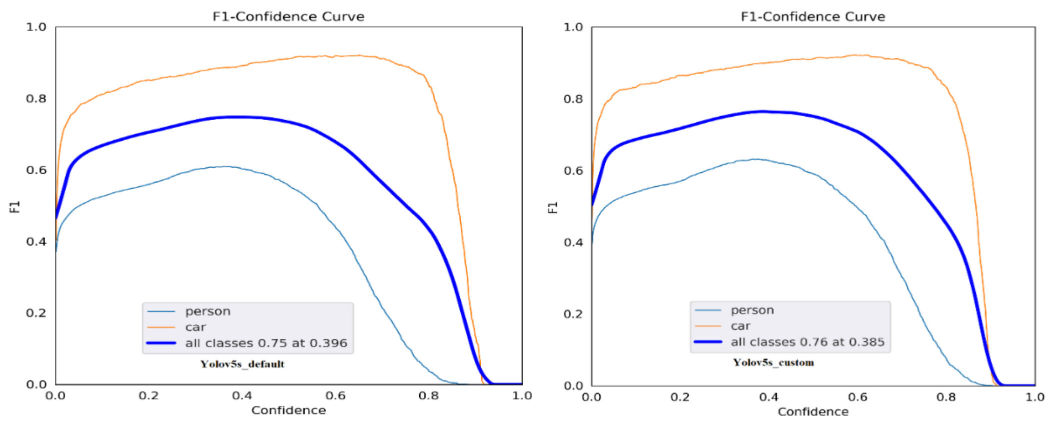


Figure 6 F1-Confidence Curve between two models

Training Performance Evaluation on ground truth data

The comparison between Yolov5s_default and Yolov5s_custom models reveals slight performance differences on the same dataset of 477 test images. The Yolov5s_custom model demonstrates a higher average recall (0.8486) compared to Yolov5s_default (0.8341), indicating better object detection capabilities. It also shows a lower average miss rate (0.1513) compared to 0.1658 for the default model. However, the custom model has a slightly higher average false positive rate (0.0125) versus the default's 0.0041. Both models exhibit high precision, with the custom model at 0.9923 and the default at 0.9890. Overall, Yolov5s_custom achieves a higher average accuracy (0.8475) compared to 0.8336 for Yolov5s_default.

Evaluation Result on Training Dataset

Experiment (Models)	**Avg. Recall**	**Avg. Miss Rate**	**Avg. False Positive Rate**	**Avg. Precision**	**Avg. Accuracy**	**Number of Test Images**
Yolov5s_default	0.8341	0.1658	0.0041	0.9890	0.8336	477
Yolov5s_custom	0.8486	0.1513	0.0125	0.9923	0.8475	477

Evaluation of YOLOv5 on Test Data

The comparison between Yolov5s_default and Yolov5s_custom models shows that the custom model performs better on a dataset of 43 test images. The Yolov5s_custom model has a higher average recall (0.5783) compared to the default model (0.5318), indicating it detects more true positives. It also has a lower average miss rate (0.4216) compared to 0.4681 for the default model, meaning fewer objects are missed. Both models have the same average false positive rate (0.0465). The custom model exhibits higher average precision (0.8798) than the default (0.8333), and a higher average accuracy (0.5781) compared to 0.5316 for the default model, indicating overall better performance.

Evaluation Result on Test Dataset

Experiment (Models)	**Avg. Recall**	**Avg. Miss Rate**	**Avg. False Positive Rate**	**Avg. Precision**	**Avg. Accuracy**	**Number of Test Images**

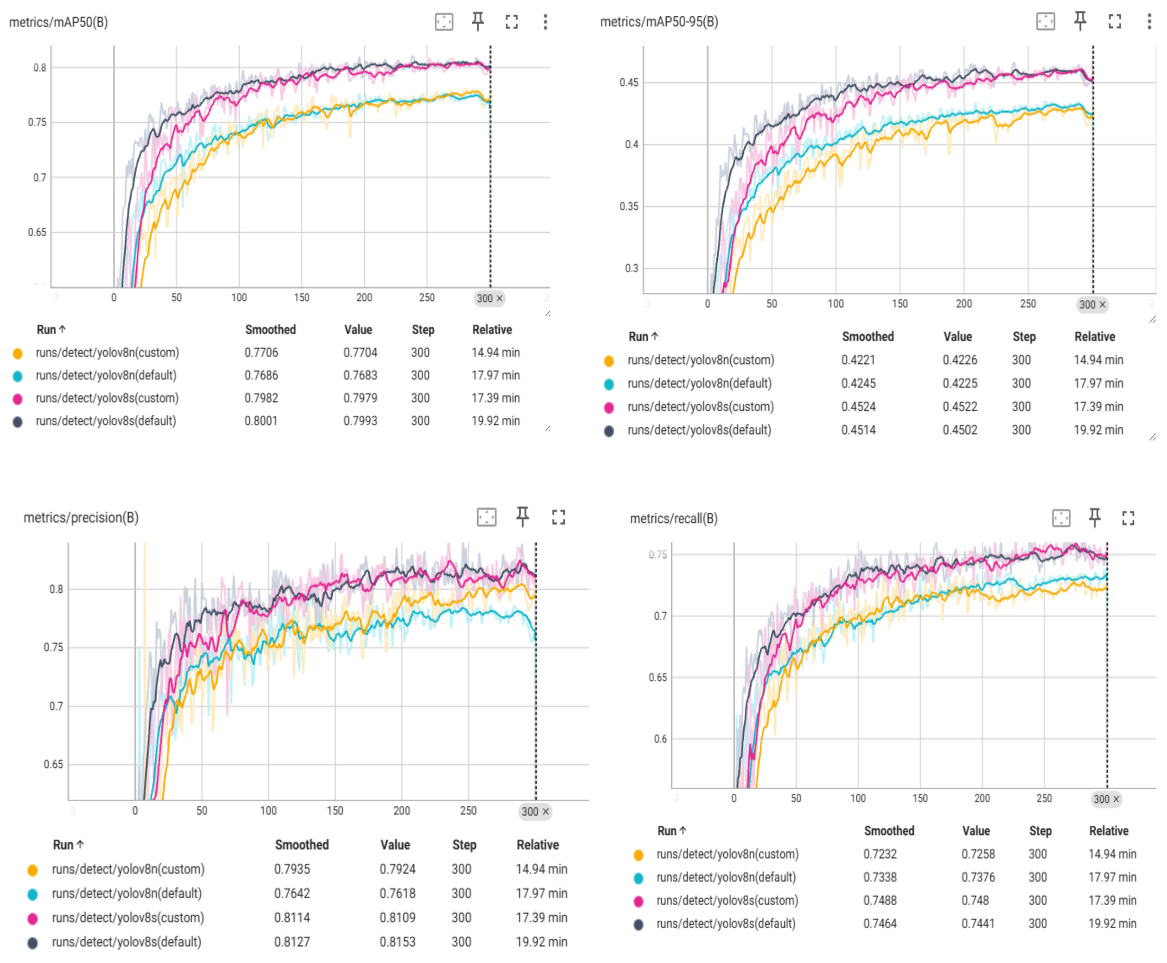
Yolov5s_default	0.5318	0.4681	0.0465	0.8333	0.5316	43
Yolov5s_custom	0.5783	0.4216	0.0465	0.8798	0.57811	43

Training Performance of YOLOv8

The results obtained by YOLOv8 after finished the training process. 1 to 300 epochs are the representation of the results obtained by the algorithms after evaluating.

Training Metrics Comparison

The provided figure displays the performance metrics for YOLOv8n and YOLOv8s models after training for 300 epochs. The metrics include mean Average Precision at 50% IoU (mAP50), mean Average Precision across different IoU thresholds (mAP50-95), precision, and recall.



metrics/mAP50(B)

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.7706	0.7704	300	14.94 min
● runs/detect/yolov8n(default)	0.7686	0.7683	300	17.97 min
● runs/detect/yolov8s(custom)	0.7982	0.7979	300	17.39 min
● runs/detect/yolov8s(default)	0.8001	0.7993	300	19.92 min

metrics/mAP50-95(B)

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.4221	0.4226	300	14.94 min
● runs/detect/yolov8n(default)	0.4245	0.4225	300	17.97 min
● runs/detect/yolov8s(custom)	0.4524	0.4522	300	17.39 min
● runs/detect/yolov8s(default)	0.4514	0.4502	300	19.92 min

metrics/precision(B)

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.7935	0.7924	300	14.94 min
● runs/detect/yolov8n(default)	0.7642	0.7618	300	17.97 min
● runs/detect/yolov8s(custom)	0.8114	0.8109	300	17.39 min
● runs/detect/yolov8s(default)	0.8127	0.8153	300	19.92 min

metrics/recall(B)

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.7232	0.7258	300	14.94 min
● runs/detect/yolov8n(default)	0.7338	0.7376	300	17.97 min
● runs/detect/yolov8s(custom)	0.7488	0.748	300	17.39 min
● runs/detect/yolov8s(default)	0.7464	0.7441	300	19.92 min

Figure All Yolov8 models matrices after training 300 epochs

1. mAP50 (Mean Average Precision at 50% IoU)

YOLOv8s shows a higher mAP50, reaching approximately 0.75 towards the end of the training. This indicates better overall performance in detecting objects with 50% overlap compared to ground truth. YOLOv8n also performs well but slightly lags behind YOLOv8s, with an mAP50 around 0.70.

2. mAP50-95 (Mean Average Precision at different IoU thresholds)

YOLOv8s again leads in this metric, with its mAP50-95 metric approaching 0.45. This suggests better performance across a range of IoU thresholds, indicating robustness in detection accuracy. YOLOv8n reaches around 0.40, which is commendable but consistently lower than YOLOv8s.

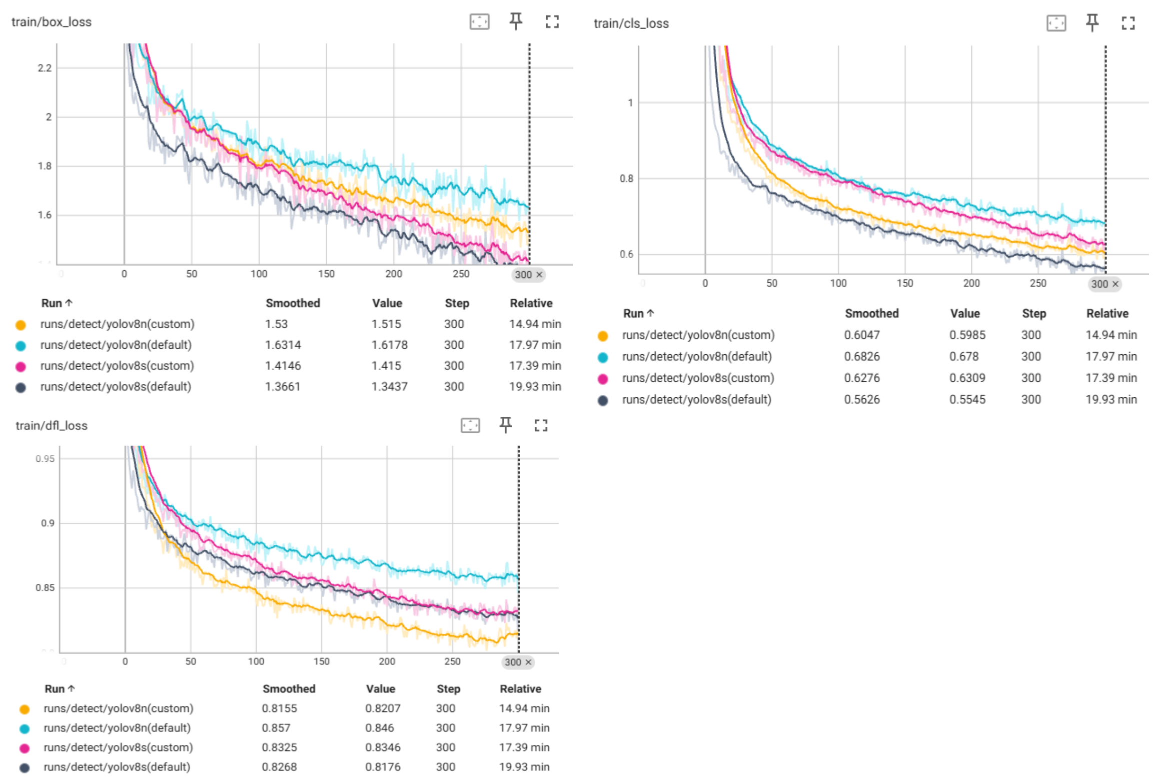
3. Precision & Recall

YOLOv8s exhibits higher precision, stabilizing at approximately 0.70. This means that a larger proportion of the detected objects are true positives. YOLOv8n shows a precision of about 0.65, indicating a slightly higher rate of false positives compared to YOLOv8s.

YOLOv8s has a recall value reaching up to 0.70, suggesting it is better at detecting most of the objects present in the images. YOLOv8n maintains a recall around 0.65, which indicates it misses more objects than YOLOv8s.

Training Loss Evaluation

The metrics evaluated include box loss, classification loss, and Distance Focal Loss (DFL), which are essential for the YOLO training process. Both models demonstrate a steady decrease in these losses, indicating effective learning and improved accuracy over time. YOLOv8s consistently exhibits lower losses than YOLOv8n, suggesting superior performance in localizing and classifying objects. Specifically, YOLOv8s maintains a box loss of around 1.35 compared to YOLOv8n's 1.40, and both models stabilize their classification loss around 0.5, with YOLOv8s slightly better. For DFL, YOLOv8s also performs better, stabilizing around 0.75. These observations highlight that YOLOv8s is more accurate and efficient in handling challenging predictions and localizations. Both models converge after approximately 150 epochs, demonstrating the effectiveness of their training



train/box_loss

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	1.53	1.515	300	14.94 min
● runs/detect/yolov8n(default)	1.6314	1.6178	300	17.97 min
● runs/detect/yolov8s(custom)	1.4146	1.415	300	17.39 min
● runs/detect/yolov8s(default)	1.3661	1.3437	300	19.93 min

train/cls_loss

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.6047	0.5985	300	14.94 min
● runs/detect/yolov8n(default)	0.6826	0.678	300	17.97 min
● runs/detect/yolov8s(custom)	0.6276	0.6309	300	17.39 min
● runs/detect/yolov8s(default)	0.5626	0.5545	300	19.93 min

train/dfl_loss

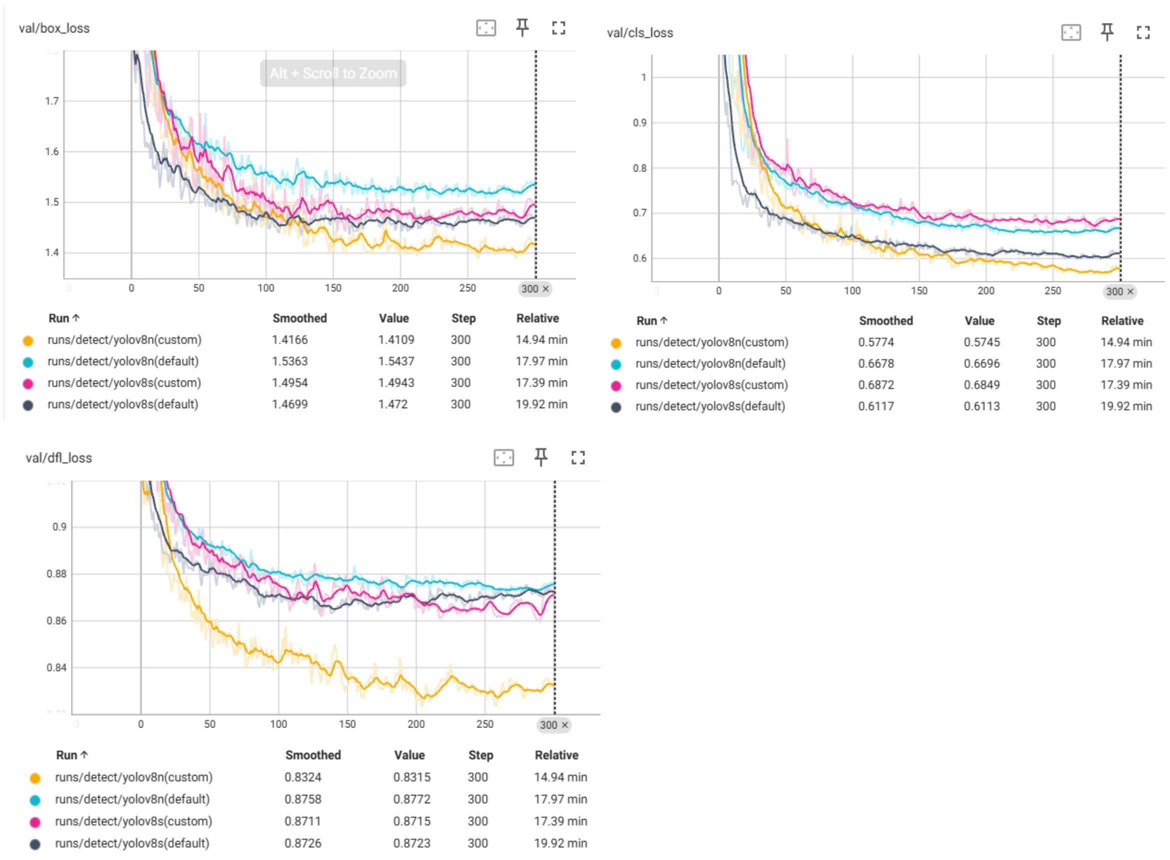
Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.8155	0.8207	300	14.94 min
● runs/detect/yolov8n(default)	0.857	0.846	300	17.97 min
● runs/detect/yolov8s(custom)	0.8325	0.8346	300	17.39 min
● runs/detect/yolov8s(default)	0.8268	0.8176	300	19.93 min

Figure 7 Training Loss of Yolov8 all models

process and generalization capabilities. The consistently lower losses for YOLOv8s across all metrics underscore its superior capability in object detection tasks compared to YOLOv8n. This analysis suggests that YOLOv8s is a more robust model, particularly for accuracy-critical applications, while YOLOv8n may still be advantageous for scenarios requiring faster inference and lower computational resources.

Validation Loss Comparison

The evaluation focuses on box loss, classification loss, and Distance Focal Loss (DFL). Both models show a decreasing trend in all three metrics, indicating improved accuracy in bounding box predictions, class label predictions, and handling difficult examples. YOLOv8s consistently demonstrates lower losses compared to YOLOv8n, with its box loss stabilizing just below 1.4, classification loss around 0.5, and DFL around 0.85. These lower loss values suggest that YOLOv8s is more accurate in localizing objects and predicting their classes, as well as more effective in focusing on challenging predictions. Both models reach stable loss values after about 50 epochs, showcasing their robust learning and generalization capabilities. The consistently lower losses of YOLOv8s across all metrics underscore its superior performance, making it a more reliable choice for object detection tasks compared to YOLOv8n.



val/box_loss

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	1.4166	1.4109	300	14.94 min
● runs/detect/yolov8n(default)	1.5363	1.5437	300	17.97 min
● runs/detect/yolov8s(custom)	1.4954	1.4943	300	17.39 min
● runs/detect/yolov8s(default)	1.4699	1.472	300	19.92 min

val/cls_loss

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.5774	0.5745	300	14.94 min
● runs/detect/yolov8n(default)	0.6678	0.6696	300	17.97 min
● runs/detect/yolov8s(custom)	0.6872	0.6849	300	17.39 min
● runs/detect/yolov8s(default)	0.6117	0.6113	300	19.92 min

val/dfl_loss

Run ↑	Smoothed	Value	Step	Relative
● runs/detect/yolov8n(custom)	0.8324	0.8315	300	14.94 min
● runs/detect/yolov8n(default)	0.8758	0.8772	300	17.97 min
● runs/detect/yolov8s(custom)	0.8711	0.8715	300	17.39 min
● runs/detect/yolov8s(default)	0.8726	0.8723	300	19.92 min

Figure 8 Validation Loss of Yolov8 all models

F1 Confidence of Each Model

The F1-Confidence curves for YOLOv8n and YOLOv8s models reveal distinct performance differences across the "person" and "car" classes, as well as their overall efficacy. For the default configurations, YOLOv8n achieves a maximum F1 score of 0.71 at a confidence level of 0.399, with the "person" class peaking at 0.85 and "car" around 0.65. The custom configuration of YOLOv8n improves, attaining an overall F1 score of 0.76 at 0.299 confidence, with "person" reaching 0.88 and "car" about 0.60. YOLOv8s shows superior performance in both configurations. In the default setting, it attains an overall F1 score of 0.78 at a confidence level of 0.253, with "person" peaking at 0.90 and "car" around 0.65. The custom configuration further enhances YOLOv8s performance, maintaining an overall F1 score of 0.78 but at a lower confidence level of 0.208, with similar peaks for "person" and "car." These results indicate that YOLOv8s outperforms YOLOv8n, particularly in the custom-trained models, demonstrating higher precision and reliability in object detection tasks.

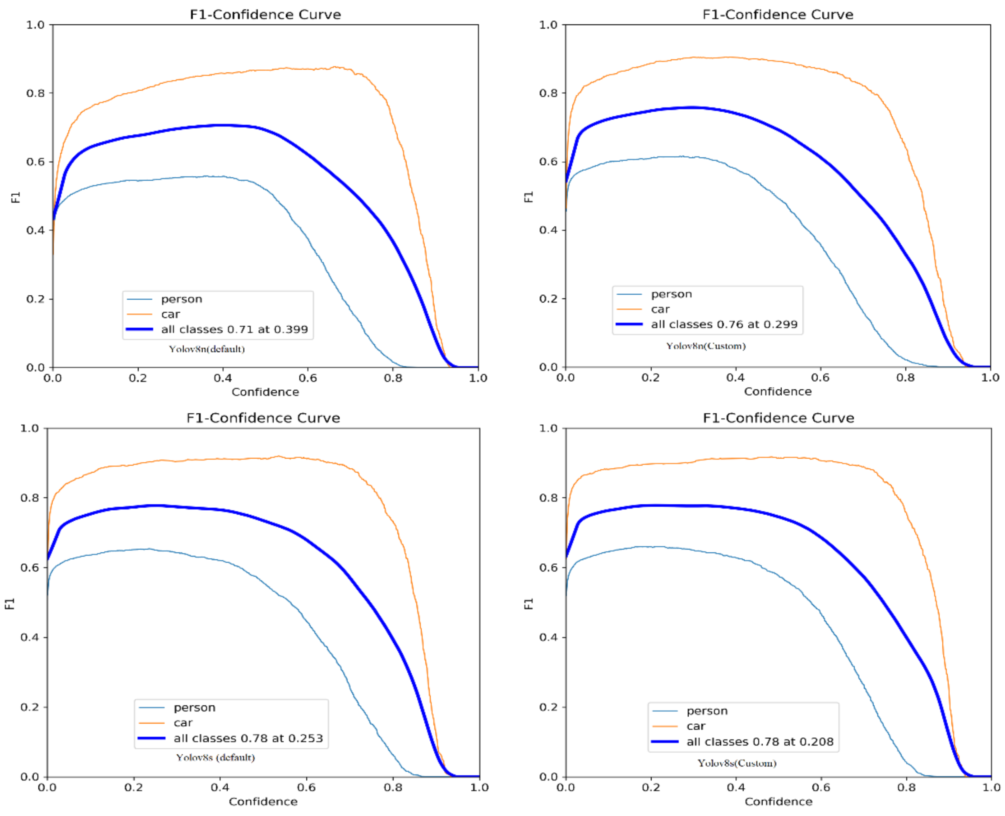


Figure 9 F1-Confidence of all Yolov8 models

Training Performance Evaluation on ground truth data:

The experimental results compare the performance of YOLOv8s and YOLOv8n models in both default and custom configurations, evaluated on 477 test images. The metrics assessed include average recall, miss rate, false positive rate, precision, and accuracy. YOLOv8s (default) demonstrates high performance with an average recall of 0.8189, a miss rate of 0.1811, zero false positives, precision of 0.9895, and an accuracy of 0.8189. The custom configuration of YOLOv8s slightly improves these metrics, achieving an average recall of 0.8237, a miss rate of 0.1763, a false positive rate of 0.0042, precision of 0.9893, and accuracy of 0.8235. In contrast, YOLOv8n (default) shows a slightly lower performance with an average recall of 0.7946, a miss rate of 0.2054, a false positive rate of 0.0021, precision of 0.9852, and accuracy of 0.7945. The custom YOLOv8n model maintains similar metrics with a slight decrease in precision to 0.9831 and accuracy to 0.7943. These results indicate that YOLOv8s outperforms YOLOv8n in both configurations, particularly in terms of recall and precision.

Evaluation Result on Training Dataset

Experiment (Models)	Avg. Recall	Avg. Miss Rate	Avg. False Positive Rate	Avg. Precision	Avg. Accuracy	Number of Test Images
yolov8s(default)	0.8189	0.1811	0.0000	0.9895	0.8189	477
yolov8n(default)	0.7946	0.2054	0.0021	0.9852	0.7945	477
yolov8s(custom)	0.8237	0.1763	0.0042	0.9893	0.8235	477
yolov8n(custom)	0.7944	0.2056	0.0021	0.9831	0.7943	477

Evaluation of YOLOv8 on Test Data

The testing results for YOLOv8s and YOLOv8n models, both in default and custom configurations were evaluated on a smaller dataset of 43 images. YOLOv8s (default) exhibited an average recall of 0.5265, a miss rate of 0.4734, a false positive rate of 0.0232, precision of 0.7906, and accuracy of 0.5265. YOLOv8n (default) showed similar performance with an average recall of 0.5213, a miss rate of 0.4786, a false positive rate of 0.0232, precision of 0.7906, and accuracy of 0.5213. The custom configuration of YOLOv8s outperformed the default, achieving an average recall of 0.5788, a miss rate of 0.4211, zero false positives, precision of 0.8372, and accuracy of 0.5788. Conversely, the custom YOLOv8n model had a lower average recall of 0.5098, a higher miss rate of 0.4901, a false positive rate of 0.0232, precision of 0.7416, and accuracy of 0.50724. These results indicate that the custom YOLOv8s model significantly enhances performance compared to its default counterpart and outperforms both configurations of YOLOv8n, particularly in terms of recall, precision, and overall accuracy.

Evaluation Result on Test Dataset

Experiment (Models)	Avg. Recall	Avg. Miss Rate	Avg. False Positive Rate	Avg. Precision	Avg. Accuracy	Number of Test Images

yolov8s(default)	0.5265	0.4734	0.0232	0.7906	0.5265	43
yolov8n(default)	0.5213	0.4786	0.0232	0.7906	0.5213	43
yolov8s(custom)	0.5788	0.4211	0.0000	0.83720	0.5788	43
yolov8n(custom)	0.5098	0.4901	0.0232	0.74160	0.50724	43

Classroom Testing Situation & Best Models Selection

Among the models evaluated, YOLOv5s_custom and YOLOv8s_custom exhibit superior performance. I have presented the YOLOv5s_custom performance for classroom testing situation where I have shown that it maintains the highest average recall (0.5788) and average precision (0.8779) on google colab GPU. Later, I switched it to my lab's Linux server.

In Linux GPU, YOLOv5s_custom achieves the highest average recall (0.8486) and average accuracy (0.8475), while also maintaining a high precision (0.9923) on the training dataset. On the test dataset, it maintains the highest average recall (0.5783) and average precision (0.8798).

YOLOv8s_custom also performs well, particularly on the test dataset, where it achieves the highest average recall (0.5788) and average accuracy (0.5788), with a perfect false positive rate (0.0000) and high precision (0.8372).

Thus, YOLOv5s_custom and YOLOv8s_custom stand out as the best-performing models in their respective evaluations.

Conclusion and Future Work

This work explores the effectiveness of deep-learning models for object detection and recognition, specifically focusing on identifying mixed instances from various content. The research evaluates these models' capabilities in discerning small and medley content and presents a theoretical and practical foundation for addressing the complex task of object detection.

Comparative analyses between YOLOv5 and YOLOv8 models reveal the superior performance of YOLOv8 in terms of accuracy and its speed while Yolov5 accurately detect tiny object effectively.

Future Work

The research on small object detection is limited, and future studies aim to improve algorithm performance by developing a customized model. We added another interference with which is called SAHI. SAHI (Slicing Aided Hyper Inference) enhances object detection by slicing input images into smaller, overlapping patches. Each patch is processed individually by the detection model, which allows for better identification of small and densely packed objects. The results from these patches are then merged to produce the final detection output, improving overall performance and accuracy. This approach ensures that even small objects, which might be missed in a full-scale image, are detected accurately. Although we have implemented this inference but it shows much misclassification detecting small content. We have to further design the model for the improvement of this model.However, the future direction of research remains uncertain. The Danish proverb "predicting the future" is a reminder of the challenges in the rapidly evolving field of technology.

References

1. Liu, W., et al. A Survey of Deep Neural Network Architectures and their Applications [J]. Neurocomputing, 2020.

2. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection [C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.

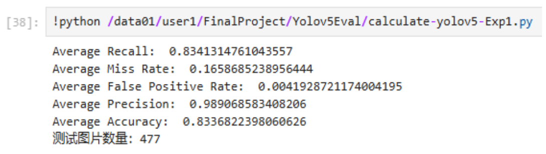
3. Zhao, Shen. YOLOv5: Understanding and Using the Improved YOLO Object Detector[J]. Applied Sciences, 2021.

4. Taylor, A., et al. Ethical and Privacy Implications of AI in Surveillance [J]. Ethics in Information Technology, 2022.

5. Roboflow Blog. YOLOv5: The Ultimate Guide [J/OL]. Available at: https://blog.roboflow.com/yolov5-the-ultimate-guide

6. Roboflow Blog. YOLOv8: The Ultimate Guide [J/OL]. Available at: https://blog.roboflow.com/yolov8-the-ultimate-guide

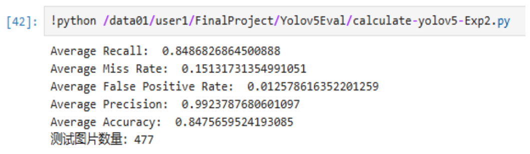
Appendix



```
[38]: !python /data01/user1/FinalProject/Yolov5Eval/calculate-yolov5-Exp1.py

      Average Recall:  0.8341314761043557
      Average Miss Rate:  0.1658685238956444
      Average False Positive Rate:  0.0041928721174004195
      Average Precision:  0.989068583408206
      Average Accuracy:  0.8336822398060626
      测试图片数量: 477
```

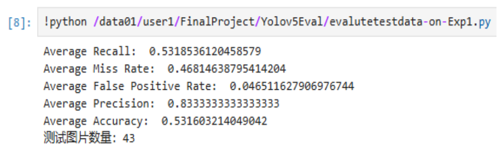
Training Evaluation Yolov5s_default



```
[42]: !python /data01/user1/FinalProject/Yolov5Eval/calculate-yolov5-Exp2.py

      Average Recall:  0.8486826864500888
      Average Miss Rate:  0.15131731354991051
      Average False Positive Rate:  0.012578616352201259
      Average Precision:  0.9923787680601097
      Average Accuracy:  0.8475659524193085
      测试图片数量: 477
```

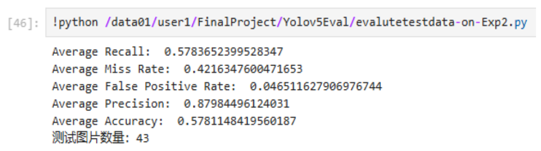
Training Evaluation Yolov5s_custom



```
[8]: !python /data01/user1/FinalProject/Yolov5Eval/evalutetestdata-on-Exp1.py

     Average Recall:  0.5318536120458579
     Average Miss Rate:  0.46814638795414204
     Average False Positive Rate:  0.046511627906976744
     Average Precision:  0.8333333333333333
     Average Accuracy:  0.531603214049042
     测试图片数量: 43
```

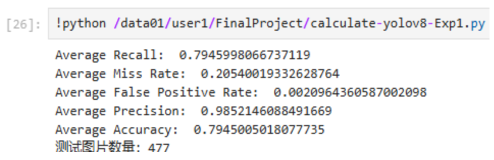
Testing Evaluation Yolov5s_default



```
[46]: !python /data01/user1/FinalProject/Yolov5Eval/evalutetestdata-on-Exp2.py

      Average Recall:  0.5783652399528347
      Average Miss Rate:  0.4216347600471653
      Average False Positive Rate:  0.046511627906976744
      Average Precision:  0.87984496124031
      Average Accuracy:  0.5781148419560187
      测试图片数量: 43
```

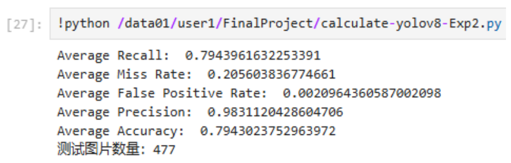
Testing Evaluation Yolov5s_custom



```
[26]: !python /data01/user1/FinalProject/calculate-yolov8-Exp1.py

      Average Recall:  0.7945998066737119
      Average Miss Rate:  0.20540019332628764
      Average False Positive Rate:  0.0020964360587002098
      Average Precision:  0.9852146088491669
      Average Accuracy:  0.7945005018077735
      测试图片数量: 477
```

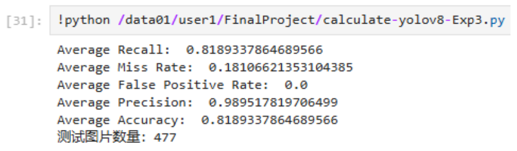
Training Evaluation Yolov8n_default



```
[27]: !python /data01/user1/FinalProject/calculate-yolov8-Exp2.py

      Average Recall:  0.7943961632253391
      Average Miss Rate:  0.205603836774661
      Average False Positive Rate:  0.0020964360587002098
      Average Precision:  0.9831120428604706
      Average Accuracy:  0.7943023752963972
      测试图片数量: 477
```

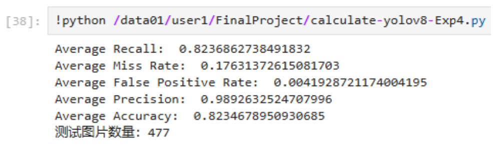
Training Evaluation Yolov8n_custom



```
[31]: !python /data01/user1/FinalProject/calculate-yolov8-Exp3.py

      Average Recall:  0.8189337864689566
      Average Miss Rate:  0.18106621353104385
      Average False Positive Rate:  0.0
      Average Precision:  0.989517819706499
      Average Accuracy:  0.8189337864689566
      测试图片数量: 477
```

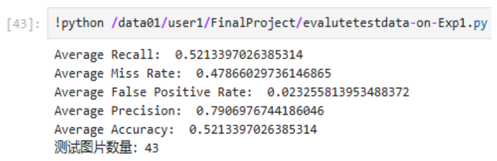
Training Evaluation Yolov8s_default



```
[38]: !python /data01/user1/FinalProject/calculate-yolov8-Exp4.py

      Average Recall:  0.8236862738491832
      Average Miss Rate:  0.17631372615081703
      Average False Positive Rate:  0.0041928721174004195
      Average Precision:  0.9892632524707996
      Average Accuracy:  0.8234678950930685
      测试图片数量: 477
```

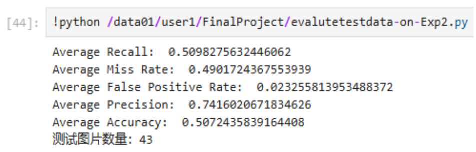
Training Evaluation Yolov8s_custom



```
[43]: !python /data01/user1/FinalProject/evalutetestdata-on-Exp1.py

      Average Recall:  0.5213397026385314
      Average Miss Rate:  0.47866029736146865
      Average False Positive Rate:  0.023255813953488372
      Average Precision:  0.7906976744186046
      Average Accuracy:  0.5213397026385314
      测试图片数量: 43
```

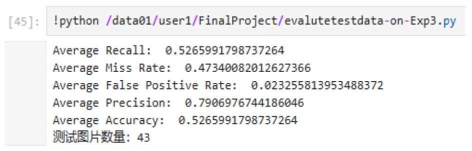
Testing Evaluation Yolov8n_default



```
[44]: !python /data01/user1/FinalProject/evalutetestdata-on-Exp2.py

      Average Recall:  0.5098275632446062
      Average Miss Rate:  0.4901724367553939
      Average False Positive Rate:  0.023255813953488372
      Average Precision:  0.7416020671834626
      Average Accuracy:  0.5072435839164408
      测试图片数量: 43
```

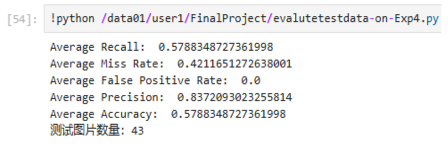
Testing Evaluation Yolov8n_custom



```
[45]: !python /data01/user1/FinalProject/evalutetestdata-on-Exp3.py

      Average Recall:  0.5265991798737264
      Average Miss Rate:  0.47340082012627366
      Average False Positive Rate:  0.023255813953488372
      Average Precision:  0.7906976744186046
      Average Accuracy:  0.5265991798737264
      测试图片数量: 43
```

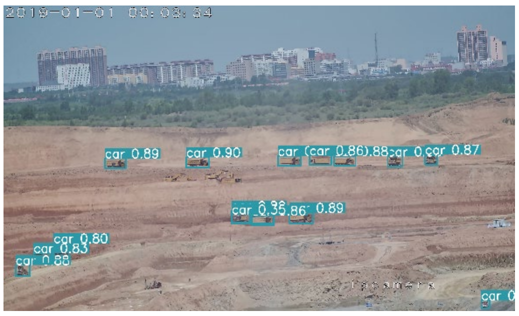
Testing Evaluation Yolov8s_default



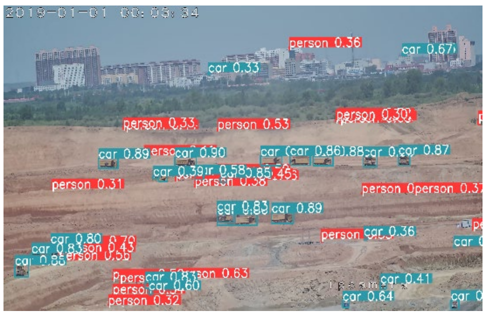
```
[54]: !python /data01/user1/FinalProject/evalutetestdata-on-Exp4.py

      Average Recall:  0.5788348727361998
      Average Miss Rate:  0.4211651272638001
      Average False Positive Rate:  0.0
      Average Precision:  0.8372093023255814
      Average Accuracy:  0.5788348727361998
      测试图片数量: 43
```

Testing Evaluation Yolov8s_custom



Before using SAHI



After using SAHI