Object Detection: *License Plate Detection using YOLO v8 and Faster-RCNN*

[*]Project Report of the course IT3320E - Introduction to Deep Learning

Ta Ngoc Minh
SID No. 20214918
Email: minh.tn214918[†]
HUST-SoICT

Tran Thanh Truong
SID No. 20214938
Email: truong.tt214938[†]
HUST-SoICT

Hoang Dinh Dung
SID No. 20214882
Email: dung.hd214882[†]
HUST-SoICT

Ngo Viet Anh
SID No. 20214875
Email: anh.nv214875[†]
HUST-SoICT

***Abstract*—This report presents a comprehensive analysis of the application of cutting-edge deep learning models, specifically YOLO v8 and Faster R-CNN, in the domain of License Plate Detection (LPD). The primary focus is on evaluating the performance, accuracy, and efficiency of these models in detecting and recognizing license plates from various types of vehicles under different environmental conditions. YOLO v8, known for its speed and real-time processing capabilities, and Faster R-CNN, renowned for its high accuracy in object detection tasks, are explored in depth. We conducted extensive experiments on diverse datasets to assess the models' robustness against challenges such as varying lighting, distances, and angles. The report also delves into the architectural nuances of each model, providing insights into why they are well-suited for LPD tasks. Additionally, we address the practical implications of deploying these models in real-world scenarios, considering factors like computational requirements and scalability. The findings of this report aim to contribute to the ongoing research in the field of computer vision and automated vehicle monitoring systems, highlighting the potential of YOLO v8 and Faster R-CNN in revolutionizing LPD technology**

I. Introduction

License Plate Detection (LDP) has been a frequent topic of research due to many practical applications, such as automatic toll collection, traffic law enforcement, private spaces access control and road traffic monitoring. This method requires higher accuracy or almost perfection. Many approaches search first for the vehicle and then its LP in order to reduce processing time and eliminate false positives.

Although LPD has been frequently addressed in the literature, many studies and solutions are still not robust enough in real-world scenarios. These solutions commonly depend on certain constraints, such as specific cameras or viewing angles, simple backgrounds, good lighting conditions, search in a fixed region, and certain types of vehicles.

The recent boost in performance for many computer vision tasks can largely be attributed to two key factors: the presence of large-scale datasets with annotations and the advent of powerful hardware, like GPUs, capable of processing vast quantities of data. In this context, Deep Learning (DL) techniques have emerged as significant players. Nonetheless, while DL approaches have made impressive strides in License Plate Detection (LPD), there remains a substantial need for LPD datasets that are specifically annotated with vehicles and license plates. The volume of training data is a critical factor in determining the effectiveness of DL methods. Higher amounts of data allow the use of more robust network architectures with more parameters and layers.

In this report, we will specifically focus on the application of advanced DL models, such as YOLO V8 and Faster R-CNN, to tackle the challenges of License Plate Detection (LPD). Each of these models brings unique strengths to the table, and our aim is to leverage these to enhance LPD accuracy and reliability. For instance, YOLO V8 is known for its exceptional speed and real-time processing capabilities, making it particularly suited for scenarios where timely detection is critical. This model's ability to rapidly process images in real-time environments makes it ideal for applications such as traffic monitoring and law enforcement. On the other hand, Faster R-CNN offers high accuracy and precision in object detection, complementing YOLO V8 by providing more detailed and precise detection in scenarios where accuracy is more critical than speed, such as in automated toll collection and parking management systems. This synergistic use of both models aims to create a comprehensive solution that addresses the various aspects and challenges of LPD

II. Dataset

After conducting a thorough search on Google for suitable datasets, we decided on one that consists of 8881 images. This dataset, available in Roboflow and downloaded in the YOLO format, is divided into three categories: 6218 images for training, 1776 for validation, and 887 for testing. It primarily features vehicles, with cars and motorbikes being the most common. Notably, the dataset includes a number of images under the class 'Light,' which represents a minority class. This dataset was chosen for its diversity, as the images were captured under a wide range of conditions, including varying angles and lighting conditions. Such diversity in the dataset is crucial for training our models to accurately detect license

[†] @sis.hust.edu.vn ; [*]Team leader.

plates in real-world scenarios, where conditions are seldom ideal and uniform. The variability in angles helps the models learn to recognize plates from different perspectives, while the variation in lighting conditions trains them to be effective under both bright and low-light situations.

Figure 1. Sample images of our dataset with LP. It shows the variety in backgrounds, lighting conditions, as well as vehicle/LP positions and types.

III. PROPOSED LPD APPROACH

In this section, we propose 2 methods namely Faster-RCNN and YOLO V8 for our problem.

A. *YOLOV8 (You Only Live Once Version 8)*

YOLOv8 [2] is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. YOLOv8 includes numerous architectural and developer experience changes and improvements over YOLOv5.

As of now, there is **no formal academic paper** available on YOLOv8, which means we don't have detailed information on the specific research methods and the comprehensive tests conducted during its development. However, by examining the available repository and the information provided on the model, we have begun to compile and document the updates and advancements incorporated into YOLOv8

1) Anchor Free Detection
   YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box.
   Anchor boxes [3] were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset.
   Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference.
2) New Convolutions
   The stem's first **6x6** conv is replaced by a **3x3**, the main building block was changed, and **C2f** replaced **C3** . The

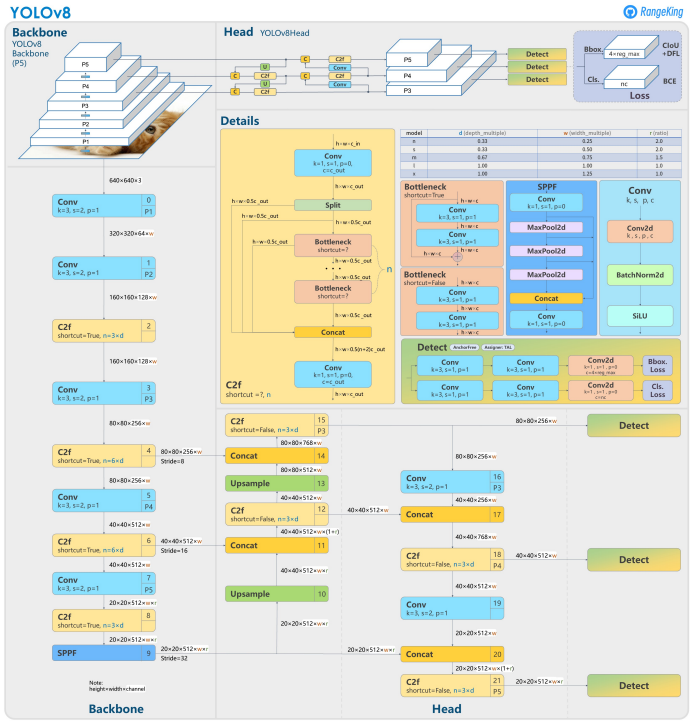

Figure 2. YOLOv8's architecture

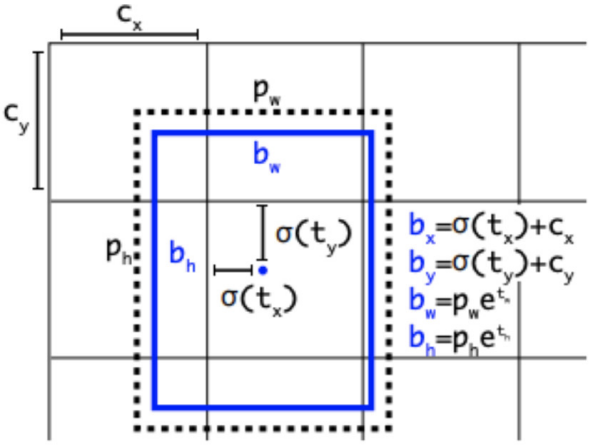

Figure 3. Visualization of an anchor box in YOLO

module is summarized in the picture below, where "f" is the number of features, "e" is the expansion rate and CBS is a block composed of a **Conv**, a **BatchNorm** and a **SiLU** later.
In **C2f**, all the outputs from the **Bottleneck** (fancy name for two 3x3 **convs** with residual connections) are concatenated. While in **C3** only the output of the last **Bottleneck** was used
The Bottleneck is the same as in YOLOv5 but the first conv's kernel size was changed from 1x1 to 3x3. From this information, we can see that YOLOv8 is starting to revert to the ResNet block defined in 2015. In the neck, features are concatenated directly without forcing the

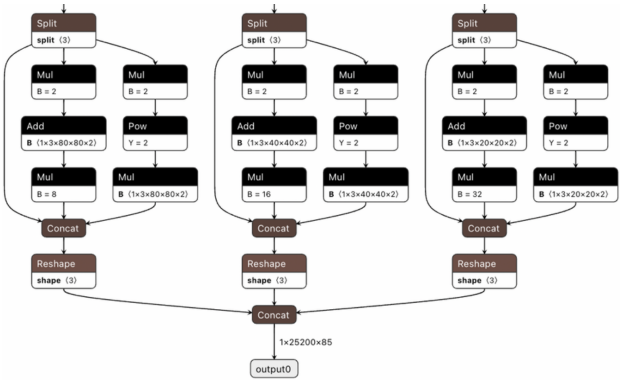

Figure 4. The detection head of YOLOv5

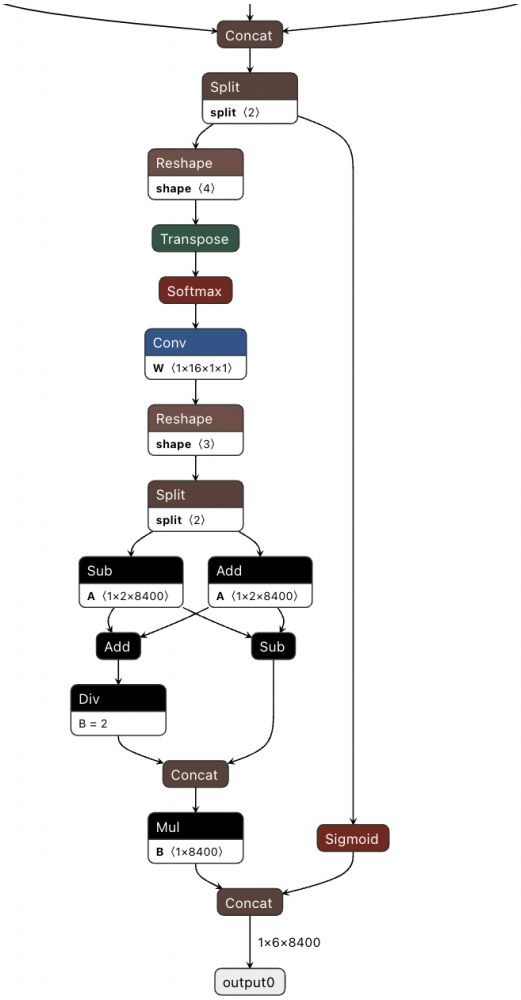

Figure 5. The detection head of YOLOv8

same channel dimensions. This reduces the parameters count and the overall size of the tensors.

3) Closing the Mosaic Augmentation
Deep learning research tends to focus on model architecture, but the training routine in YOLOv5 and YOLOv8 is an essential part of their success. YOLOv8 augments

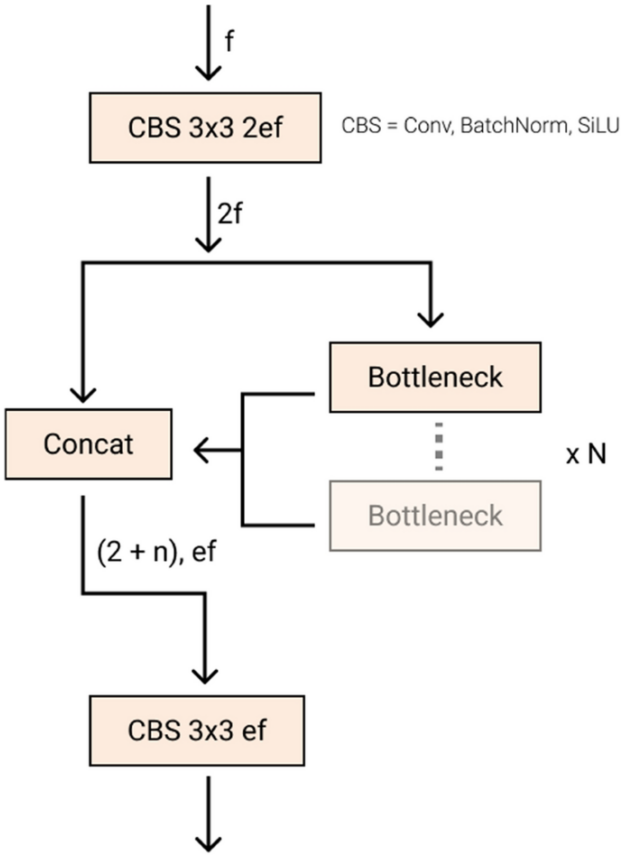

Figure 6. New YOLOv8 C2f module

images during training online. At each epoch, the model sees a slightly different variation of the images it has been provided. One of those augmentations is called mosaic augmentation[**??**]. This involves stitching four images together, forcing the model to learn objects in new locations, in partial occlusion, and against different surrounding pixels.
However, this augmentation is empirically shown to degrade performance if performed through the whole training routine. It is advantageous to turn it off for the last ten training epochs.
This sort of change is exemplary of the careful attention YOLO modeling has been given in overtime in the YOLOv5 repo and in the YOLOv8 research.

4) YOLOv8 Accuracy Improvements
YOLOv8 research was primarily motivated by empirical evaluation on the COCO benchmark. As each piece of the network and training routine are tweaked, new experiments are run to validate the changes effect on COCO modeling. COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds. The image below shows the accuracy

of YOLOv8 on COCO, using data collected by the Ultralytics team

▼ Detection

Model	size (pixels)	mAP^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset. Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance. Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

Figure 7. YOLOv8 COCO evaluation

B. *Faster R-CNN*

Faster R-CNN stands as a top-tier framework in object detection, boasting extensive integrations despite its relative age. While its name suggests speed, it is actually slower compared to models like YOLOv3 or MobileNet, but offers enhanced accuracy. TensorFlow Object Detection API readily includes it among its default architectures, complete with pre-trained weights. The implementation discussed in this notebook seamlessly connects with Tensorboard, leveraging its capabilities for more efficient object detection applications.

1) Faster R-CNN Architecture
The Faster R-CNN utilizes is a two-stage deep learning object detector: first, it identifies regions of interest and then passes these regions to a convolutional neural network. The outputted feature maps are passed to a support vector machine (SVM) for classification. Regression between predicted bounding boxes and ground truth bounding boxes is computed. Below is the general architecture for the Faster R-CNN.

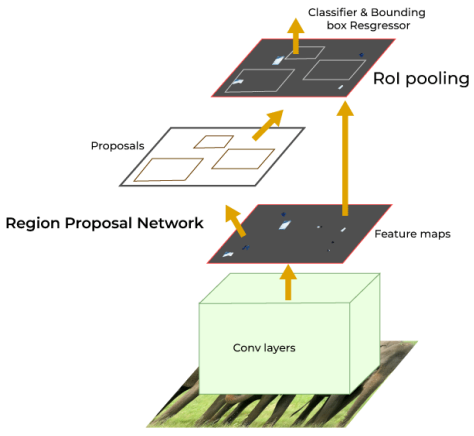

Figure 8. General Architecture of Faster R-CNN

2) Region Proposal Networks
A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score.1 We model this process with a fully convolutional network, which we describe in this section. Because our ultimate goal is to share computation with a Fast R-CNN object detection network, we assume that both nets share a common set of conv layers. To generate region proposals, we slide a small network over the conv feature map output by the last shared conv layer. This network is fully connected to an n × n spatial window of the input conv feature map. Each sliding window is mapped to a lower-dimensional vector. This vector is fed into two sibling fully connected layers—a box-regression layer (reg) and a box classification layer (cls). Note that because the mini-network operates in a sliding-window fashion, the fully connected layers are shared across all spatial locations. This architecture is naturally implemented with an n × n conv layer followed by two siblings 1 × 1 conv layers (for reg and cls, respectively). ReLUs are applied to the output of the n × n conv layer

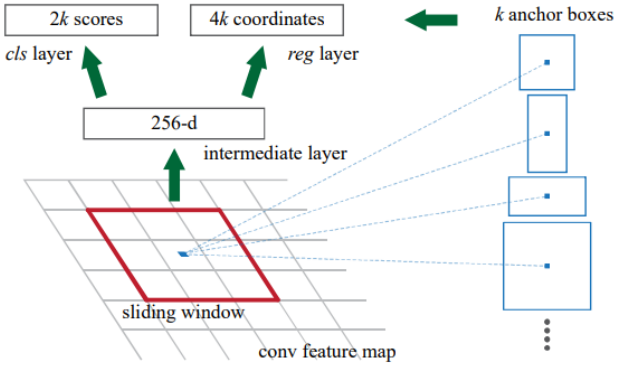

Figure 9. Region Proposal Network (RPN)

3) Translation-Invariant Anchors
At each sliding-window location, we simultaneously predict k region proposals, so the reg layer has 4k outputs encoding the coordinates of k boxes. The cls layer outputs 2k scores that estimate probability of object / not-object for each proposal. The k proposals are parameterized relative to k reference boxes, called anchors. Each anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio. We use 3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position. For a conv feature map of a size W ×H (typically 2,400), there are W Hk anchors in total. An important property of our approach is that it is translation invariant, both in terms of the anchors and the functions that compute proposals relative to the anchors. As a comparison, the MultiBox method uses k-means to generate 800 anchors, which are not translation invariant. If one translates an object in an image, the proposal

should translate and the same function should be able to predict the proposal in either location. Moreover, because the MultiBox anchors are not translation invariant, it requires a (4+1)×800-dimensional output layer, whereas our method requires a (4+2)×9-dimensional output layer. Our proposal layers have an order of magnitude fewer parameters and thus have less risk of overfitting on small datasets, like PASCAL VOC.

IV. Model Evaluation

In this section, we evaluate the performance of the object detection models trained on our dataset. Various metrics such as recall, precision, mean Average Precision (mAP), and loss values are used to quantify the effectiveness of the models under different conditions, such as with and without data augmentation, and with pretraining versus no pretraining.

A. Model Tuning and Configuration Analysis

The model tuning phase in our project is both critical and insightful. We employed an iterative approach to refine our object detection models, which involved experimenting with various permutations and combinations of data augmentation and the use of pretrained models. Below, we detail the configurations used and the rationale behind each:

- **No Augmentation, No Pretraining (no_augment_no_pretrained)**: This fundamental setup is often considered as a control scenario. Here, the model is trained purely based on the raw dataset, without any external influences such as data augmentation or prior knowledge from pretrained models. This configuration establishes a performance baseline, which is crucial for comparative analysis.
- **Augmentation, No Pretraining (augment_no_pretrained)**: Data augmentation is a powerful technique to artificially expand the training dataset by applying various transformations to the original images. These transformations include scaling, cropping, flipping, and color perturbation. By introducing these variations, the model is expected to learn more robust features that are invariant to such changes, thereby improving its ability to generalize from limited data.
- **No Augmentation, Pretrained (no_augment_pretrained)**: Starting the training process from pretrained weights is a form of transfer learning. This method relies on the premise that models pretrained on large and diverse datasets (such as ImageNet) have learned a rich set of features that can be effectively transferred to a new task with minimal adaptation. This approach is especially beneficial when the target dataset is relatively small, as it can help the model converge faster and with potentially higher accuracy.
- **Augmentation, Pretrained (augment_pretrained)**: Combining data augmentation with a pretrained model aims to harness the combined power of enhanced dataset variability and learned feature representations. This dual approach is hypothesized to lead to a model that not only converges quickly but also exhibits superior performance on unseen data, thanks to its extensive exposure to varied training examples and sophisticated feature extraction capabilities.

For each configuration, we meticulously monitor a spectrum of metrics. Recall measures the model's ability to detect all relevant instances, precision reflects its accuracy in these detections, and the mean Average Precision (mAP) provides an aggregated measure of both across different object detection thresholds. Furthermore, we scrutinize the loss metrics—classification loss, box regression loss, and overall loss—throughout the training and validation cycles to understand the learning trajectory and to identify any signs of underfitting or overfitting.

The graphical representations that follow not only exhibit the trends and patterns in these metrics over training epochs but also shed light on the trade-offs inherent to each configuration. For instance, models without data augmentation may overfit, as indicated by a divergence between training and validation loss. On the other hand, models with excessive augmentation might underperform if the augmented features do not represent the test data distribution accurately. Similarly, while pretrained models may demonstrate a strong start, the gains may plateau if the transfer learning does not align well with the task-specific nuances. These considerations drive our subsequent model refinement strategies and ultimately inform our decision on the best model to deploy for real-world applications.

B. Analytical Assessment of Model Metrics

Our analysis leverages a suite of detection metrics, each providing a different perspective on the model's performance.

1) Recall Analysis: Recall measures the model's sensitivity to detecting instances of objects across the dataset. High recall values indicate a model's strength in covering the full spectrum of object instances, reducing the likelihood of missed detections.

2) Precision Analysis: Precision quantifies the exactness of the model in predicting object instances, with high precision pointing to a lower rate of false positives—a critical attribute in precision-critical applications.

3) Mean Average Precision (mAP) Analysis: The mAP metric provides an aggregated measure of precision and recall across multiple IoU thresholds, offering a balanced evaluation of the model's overall prediction accuracy and localization ability.

mAP50-95: mAP at IoU thresholds ranging from 0.5 to 0.95 reflects a comprehensive model assessment, encapsulating a broad range of detection difficulties.

mAP50: mAP at an IoU threshold of 0.5 is indicative of the model's performance under less restrictive conditions, providing an additional layer of insight into its detection capabilities.

C. Loss Metrics Dissection

We further dissect the model's learning trajectory by examining various loss components, which serve as indicators of the model's predictive discrepancy.

1) Validation Loss: Validation loss offers an indicator of model generalizability, with a lower loss on validation data being indicative of better model performance on unseen data.

2) Component-wise Validation Loss: The validation loss is further broken down into its constituent parts, each corresponding to a specific aspect of the detection task: directional field loss, classification loss, and box regression loss.

3) Training Loss: The training loss is indicative of the model's learning progression, with a consistently decreasing loss suggestive of the model's increasing adeptness at capturing the training data distribution.

Through this analytical lens, our models were meticulously evaluated to ensure that the final model selection would align with the high standards required for deployment in real-world scenarios. The plots presented above are a testament to the rigorous evaluation framework adopted in our study.

V. Implementation

All the sources code and implementation are here: Repository.

VI. Summaries and Improvements

After running several times, we get the score of configurations represented in the table below. The numbers in each cell are mAP50-95, mAP50.

	Model		
	Measurement	**YOLOv8**	**Faster-RCNN**
NoA&NoP	mAP50-95	0.43	0.323
	mAP50	0.747	0.484
A&NoP	mAP50-95	0.512	0.322
	mAP50	0.829	0.484
NoA&P	mAP50-95	0.616	0.321
	mAP50	0.935	0.483
A&P	mAP50-95	0.617	0.326
	mAP50	0.951	0.483

*NoA: No Augmentation
*A: Augmentation
*NoP: No pretraining
*P: Pretrained

References

[1] Piotr Skalski(2023). How to Train YOLOv8 Object Detection on a Custom Dataset.https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/

[2] Ultralytics YOLOv8 Documentations.https://docs.ultralytics.com/

[3] Roboflow. YOLOv8: A New State-of-the-Art Computer Vision Model. https://yolov8.com/

[4] Shaoqing Ren, Kaiming He, Ross Girshick & Jian Sun(2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[5] M.Lokanath, K.Sai Kumar & E.Sanath Keerthi(2017). Accurate object classification and detection by faster-RCNN.

[6] Raducu Gavrilescu, Cristian Zet, Cristian Foșalău, Marcin Skoczylas, David Cotovanu(2018).Faster R-CNN:an Approach to Real-Time Object Detection.

[7] Raducu Gavrilescu, Cristian Zet, Cristian Fosalau, Marcin Skoczylas(2018).Faster R-CNN:an Approach to Real-Time Object Detection.

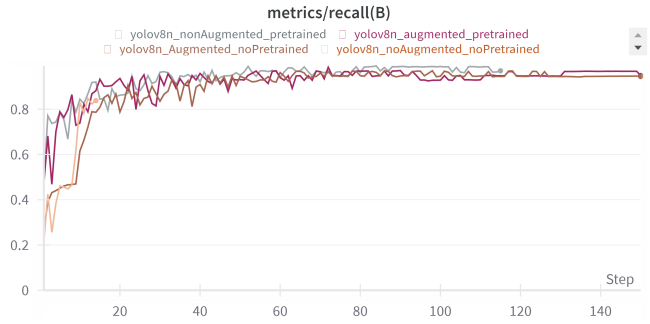

Figure 10. Recall trends across different model configurations over training epochs.

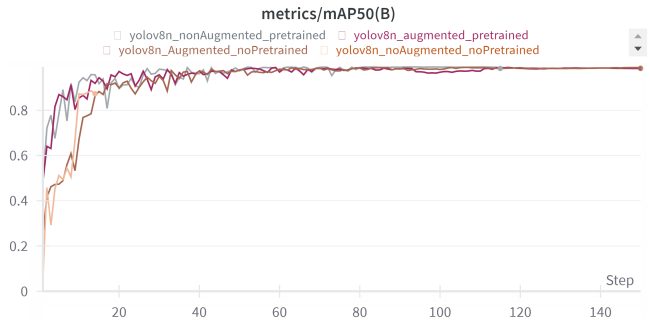

Figure 13. mAP50 values offering a view of the model's performance at a fundamental level of object localization.

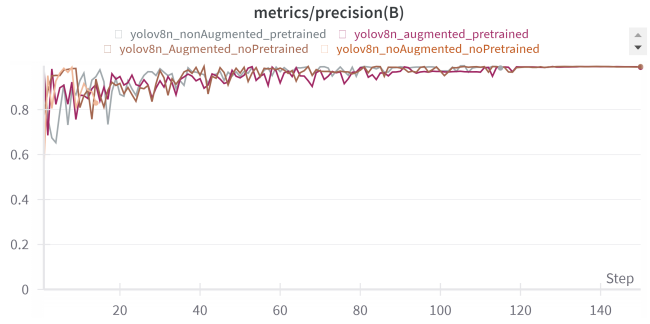

Figure 11. Precision trends highlighting the model's accuracy in object prediction.

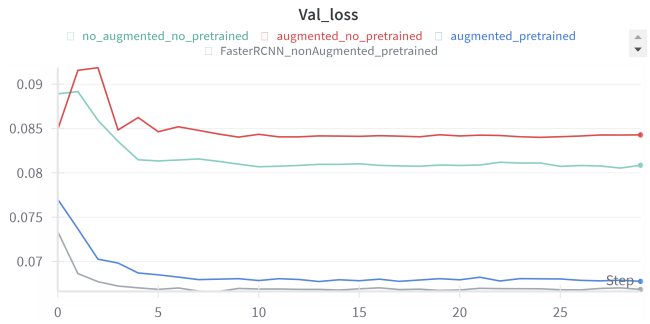

Figure 14. Comparison of validation loss between the YOLO configurations and a baseline Faster R-CNN model.

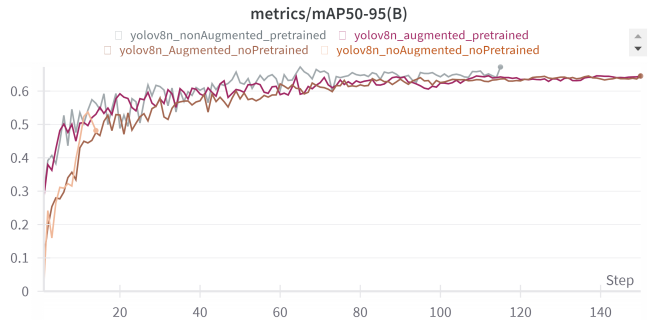

Figure 12. mAP50-95 values representing the model's averaged precision across varying IoU thresholds.

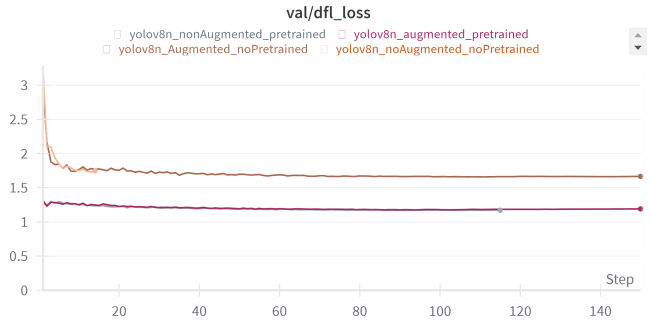

Figure 15. Directional field loss trends, indicating the model's accuracy in object boundary localization.

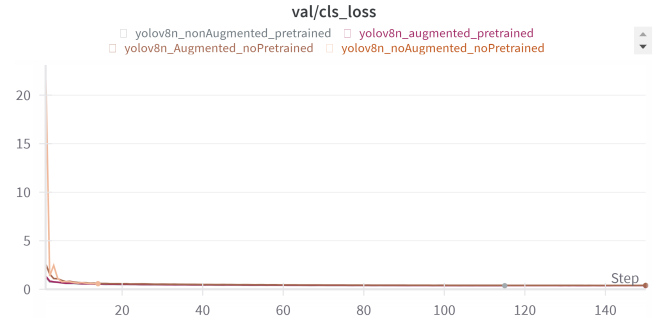

Figure 16. Classification loss trends, providing insight into the model's object classification performance.

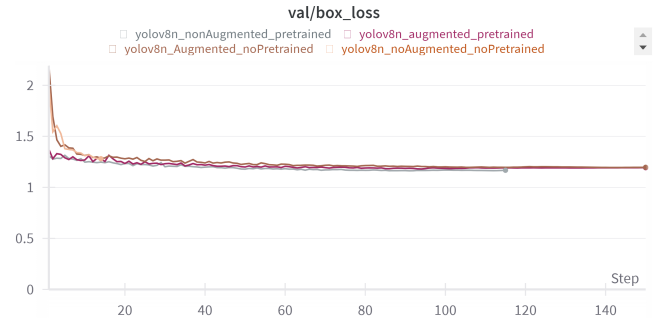

Figure 17. Box regression loss trends, reflecting the model's precision in predicting the extent of object bounding boxes.

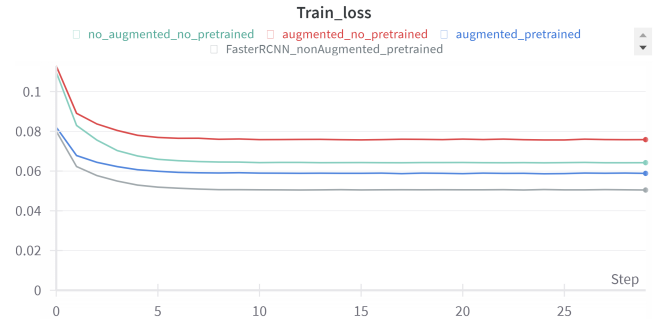

Figure 18. Training loss trends for the YOLO model in comparison to the Faster R-CNN model.