iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IX **Month of publication:** September 2025

DOI: https://doi.org/10.22214/ijraset.2025.74260

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

Performance Evaluation of YOLOv8 for Object Detection on the COCO Dataset

Shital Katkade[1], Dr. Ramesh R. Manza[2], Priya Shinde[3], Sujata Ambhore[4], Reema Lahane[5]
Department of Computer Science & IT, Dr. Babasaheb Ambedkar Marathwada University, Chhatrapati Sambhajinagar, India

Abstract: ***In computer vision, object detection is a crucial task with many applications, such as robots, autonomous cars, and surveillance. The newest model in the YOLO (You Only Look Once) family, YOLOv8, brings important architectural enhancements to improve model efficiency, speed, and detection accuracy. YOLOv8 is a small object detection framework based on deep learning that has been used a lot in computer vision tools. Building blocks, preparation, and data addition methods are used to make it work. The training process was carefully set up to work with the limited resources that were available while still using modern deep learning techniques. The model was then refined on the custom dataset that was made just for the job. In terms of precision, recall, and class-wise detection ability, the results show some good signs. Our findings show that YOLOv8 is appropriate for both cloud-based and embedded applications since it maintains real-time inference speeds while achieving excellent detection accuracy. The results demonstrate YOLOv8's potential as an effective tool for a variety of real-world object detection applications.***

Keywords: YOLO, Object Detection, COCO, training, testing, Deep Learning

I. INTRODUCTION

Object detection is the process of finding things in a picture or video frame. It is a basic computer vision job. It is necessary for many things, like smart shopping, medical imaging, monitoring, and self-driving cars [1, 2]. Traditional object recognition methods, such as Haar cascades and Histogram of Orientated Gradients (HOG), were not as flexible or accurate in complex situations because they relied too much on features that had to be made by hand [3]. Using deep learning has made object recognition much more accurate and efficient, especially for convolutional neural networks (CNNs) that can find objects of all shapes, sizes, and positions [4]. Object detection has come a long way thanks to systems based on deep learning. When region-based CNNs (R-CNNs) came up with region proposal methods to improve detection accuracy [5], Fast R-CNN and Faster R-CNN made computing more efficient. In any case, these methods weren't good for real-time situations and took a lot of time to compute. Since area proposal networks were no longer needed, single-stage detectors like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) could be used instead, focussing on speed and efficiency [6]. By showing object recognition as a regression problem, YOLO changed the way things were thought about in a way that made inference much faster while keeping accuracy at a competitive level. Several performance-related changes have been made to YOLO over the years, such as better feature extraction, better anchor box methods, and the addition of attention processes to help with object positioning [7–9], [10].

In 2016, Joseph Redmon and colleagues originally presented YOLO, a real-time object detection framework that approached detection as a single regression problem. YOLO is substantially faster than region-based methods, which carry out several region proposals, because it predicts bounding boxes and class probabilities from an input image in a single forward pass. In real-time situations, YOLOv1 outperformed earlier techniques thanks to its remarkable speed-accuracy balance [11]. Accuracy, generalization, and efficiency were enhanced in later YOLO versions. Batch normalization and anchor boxes were included in YOLOv2, and Darknet-53 was added as a backbone for improved feature extraction in YOLOv3 [12]. Due to its increased accuracy and computational efficiency, YOLOv4 and YOLOv5 have become widely used in industrial applications [13], [14].

Every iteration of YOLO has been improved to increase computing performance, accuracy, and efficiency.

Single-stage object detection was introduced by YOLOv1 (2016), which performed in real-time but had trouble with small objects.

1) YOLOv2 (YOLO9000) (2017): Better detection and classification with the use of higher-resolution photos, batch normalization, and anchor boxes.
2) YOLOv3 (2018): Presented multi-scale feature detection and a deeper backbone network (Darknet-53)[12].
3) YOLOv4 (2020): CSPDarkNet was used to optimize the network topology, and improvements including Mish activation and mosaic data augmentation were made.

©IJRASET: All Rights are Reserved

4) YOLOv5 (2020): Emphasized deployment optimization, usability, and ease of training. Increased accuracy and efficiency in YOLOv6 and YOLOv7 (2022), with YOLOv7 emerging as the quickest real-time detector[15],[16]. For enhanced real-time performance, YOLOv8 (2023) added a new backbone, sophisticated anchor-free detection, and increased model efficiency [17]. In order to assess YOLOv8's performance and identify its advantages and disadvantages, this study will compare it to earlier iterations, namely YOLOv5, YOLOv6, and YOLOv7. The comparative study will concentrate on:
5) Accuracy: Assessing detection performance with common measures for object detection, like mean average precision, or mAP [11], [14]-[16].
6) Inference Speed: Assessing processing speed in real time across various hardware setups [14]-[16].
7) Computational Efficiency: Evaluating inference delay, training time, and model size [15], [16].
8) Real-world Uses: Examining situations in which YOLOv8 performs better than its predecessors [18].

This study sheds light on the development of YOLO and its suitability for use in contemporary computer vision applications by examining these variables. The findings will help developers and researchers choose the best YOLO variant for their particular use cases [19].

II. METHODOLOGY

A. Dataset

For training and testing YOLOv8, the COCO (Common Objects in Context) dataset was used. A lot of people use it as a benchmark for tasks like object recognition, segmentation, and keypoint detection. COCO has more than 200,000 labelled images of objects from 80 different groups. It is a challenging and varied dataset for training deep learning models. The dataset has items of different sizes, shapes, and levels of visibility, so it can be used in a wide range of real-life situations. Its wide range of uses makes it a great choice for testing object recognition models because it is so similar to real-life situations.

The following factors led to the selection of the COCO dataset for training and assessment:

1) Diversity: The dataset is a thorough benchmark since it contains items in a broad range of settings, lighting scenarios, and occlusions.
2) Challenging Object Categories: The dataset tests the detection models' resilience by containing small, medium, and large objects.
3) Measures for Standardized Evaluation: COCO offers established measures like as mean Average Precision (mAP) that allow for equitable comparisons across various object detection algorithms.
4) Real-World Applications: Because the dataset closely reflects real-world situations, models developed on COCO are very generalizable to a wide range of applications, including robots, autonomous driving, and surveillance.

B. Image Preprocessing and Data Augmentation

Several methods for preprocessing were used to make the model more general. One important step in the preprocessing process was resizing the pictures. All of the input images were made 640 × 640 pixels so that they would work with YOLOv8 [18]. By making sure that all samples have the same input size, this standardisation helps training and inference go more smoothly.

To further enhance model robustness, various data augmentation techniques were applied to the training images that are:

1) Horizontal Flipping: The images were flipped left-to-right with a 50% probability. This augmentation helps the model recognize objects regardless of their orientation [13], [18].
2) Random Rotation: The images were rotated randomly within a range of ±10 degrees to introduce variations in object positioning [13], [18].
3) Gaussian Noise: Small amounts of noise were added to the images to simulate real-world variations and improve generalization [20].
4) Brightness & Contrast Adjustment: The brightness and contrast were randomly adjusted to handle varying lighting conditions [13], [18].
5) Scaling and Cropping: Random scaling and cropping ensured the model could detect objects at different distances [13], [18].
6) Applying these augmentation techniques helps the model learn robust features, reducing overfitting and improving performance on unseen images. Below are sample images illustrating different augmentation techniques:

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

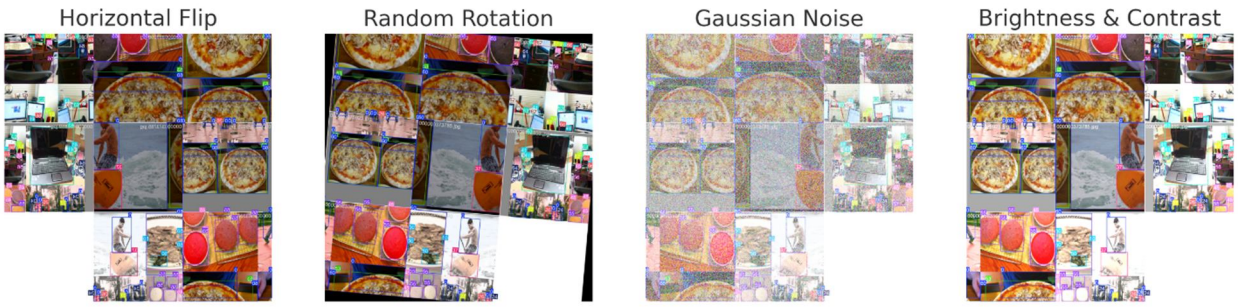


Fig. 1 Preprocessed images

Through these preprocessing steps, YOLOv8 was trained more effectively, improving its ability to detect objects under diverse conditions [18].

C. Model Training

The YOLOv8n (nano) model was used to find objects in this study because it was the best mix of speed, accuracy, and computer efficiency [18], [13]. YOLOv8n is the lightest model in the YOLOv8 family. It works especially well for apps on platforms with few resources, like those that only use CPU hardware or embedded systems like Raspberry Pi [21, 22]. Even though it has a small design, it benefits from the advanced design features of the YOLOv8 series, such as anchor-free detection, disconnected detection heads, and an improved neural backbone, which all make it better at a number of detection tasks [18].

Transfer learning was used to speed up model convergence and improve beginning performance. This was done by starting the model with weights that had already been trained on the COCO dataset [11], [23]. This method lets the model keep generalised visual features from a big and varied dataset. It is then tweaked on a custom dataset made just for the job. The training process had five epochs, which was a good number for testing the dataset integration and looking at how the machine learnt early on [24]. A 416x416 pixel input image resolution was used to keep the training pipeline consistent and to make sure it would work with the model's design [18], [25]. Because CPU-based training environments have their limits, a batch size of 8 was chosen to find a good mix between memory needs and learning stability [24].

As part of the training process, the YOLOv8 framework [18] offered an automatic optimiser selection mechanism that was used for optimisation. When "auto" is selected, the framework doesn't use the learning rates and momentum values set by the user. Instead, it figures out the best choices automatically based on the dataset and model properties. So, AdamW, a type of the Adam optimiser known for its separated weight decay that makes generalisation and training stability better [26],[27] was chosen as the optimiser.

The automatically chosen hyperparameters included:

1) A learning rate of 0.000119, fine-tuned for the model's learning capacity.
2) A momentum value of 0.9, which helps accelerate learning and reduce oscillation.
3) Three distinct parameter groups for weight decay:
    - Parameters such as normalization layers with no weight decay,
    - Convolutional weights with a decay of **0.0005** to prevent overfitting,
    - Bias parameters, which were also excluded from decay.

These optimizer configurations allowed for more nuanced control over the learning process, helping to prevent over-regularization of certain layers while still promoting generalization across the network.

The training was conducted on a CPU with no parallel data-loading workers, which limited data throughput but ensured compatibility with the available hardware. To mitigate the I/O bottleneck during data fetching, image caching was enabled. This allowed images to be preloaded into memory, reducing loading times during each epoch.

Furthermore, the training process included systematic logging and checkpointing. Model weights were saved periodically at the end of each epoch, enabling the tracking of performance progression and allowing for model restoration or re-evaluation at intermediate stages.

In summary, the model training phase was carefully configured to align with resource limitations while still utilizing modern deep learning practices. This included leveraging transfer learning, adaptive optimization strategies, and efficient data handling techniques. These design decisions laid a robust foundation for further experimentation, fine-tuning, and performance benchmarking in the context of object detection using YOLOv8.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

III. RESULTS AND DISCUSSION

The object detection model was trained using the YOLOv8n architecture over 5 epochs on a custom subset of the COCO dataset. During training, extensive data augmentation techniques were applied to enhance model generalization and robustness. These included mosaic augmentation, random scaling, flipping, and color jittering. The impact of these augmentations is visually evident in the training samples presented in train_batch0.jpg and train_batch1.jpg, which display the transformed input images along with the corresponding ground truth bounding boxes. These images confirm that the dataset preparation and augmentation strategies were effective in diversifying the visual input space, which is critical for robust detection in real-world outdoor environments.
The following images shows the training batch outputs:



Fig.2 train_batch0



Fig.3 train_batch1

A. Validation Results

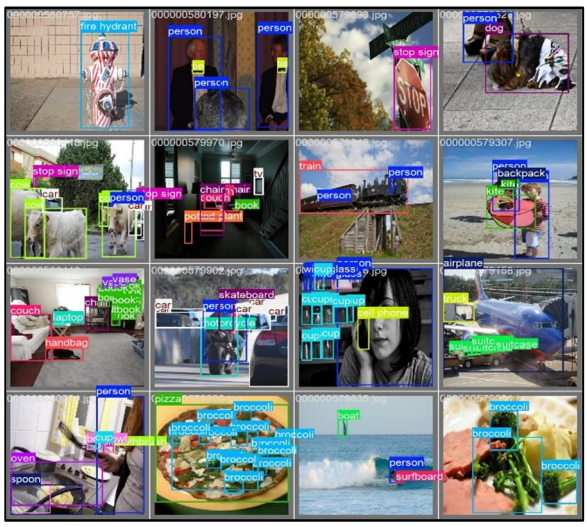


Fig.4 Ground Truth Labels: val_batch0_labels

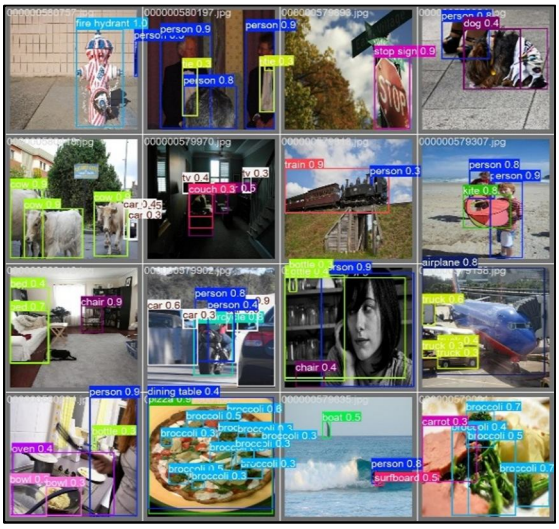


Fig.5 Model Predictions: val_batch0_pred

After training, the performance of the model was evaluated using multiple metrics. The results.png graph illustrates the trends in training loss, precision, recall, and mean Average Precision (mAP@0.5) across epochs. A gradual decrease in loss and improvement in precision and recall indicate that the model was effectively learning the object features.
Further evaluation using per-class metrics was conducted. The Precision **(**P_curve.png**)**, Recall **(**R_curve.png**)**, F1-score **(**F1_curve.png**)**, and Precision-Recall curve **(**PR_curve.png**)** offer a detailed analysis of the model's ability to correctly classify objects. The F1 curve peaks around certain classes, indicating a balanced performance in terms of both precision and recall. Similarly, the PR curve confirms a favorable trade-off between precision and recall across different thresholds.

The confusion matrix (confusion_matrix.png) provides insights into the classification accuracy across various object classes, while the normalized confusion matrix(confusion_matrix_normalized.png) shows the distribution of correct and incorrect predictions in percentage form [28]. Most of the diagonal elements show high intensity, indicating correct predictions with relatively fewer misclassifications. Additionally, the class distribution heatmap **(**labels.jpg**)** and the label correlogram(labels_correlogram.jpg) reveal how object instances are distributed across classes and their correlations. This helps in understanding potential class imbalances and co-occurrence, which are important for refining future training strategies. Overall, despite being trained for only 5 epochs on CPU with a lightweight YOLOv8n model, the results show promising trends in terms of precision, recall, and class-wise detection performance. This validates the capability of the YOLOv8 framework in performing efficient object detection, even with limited computational resources and training time.

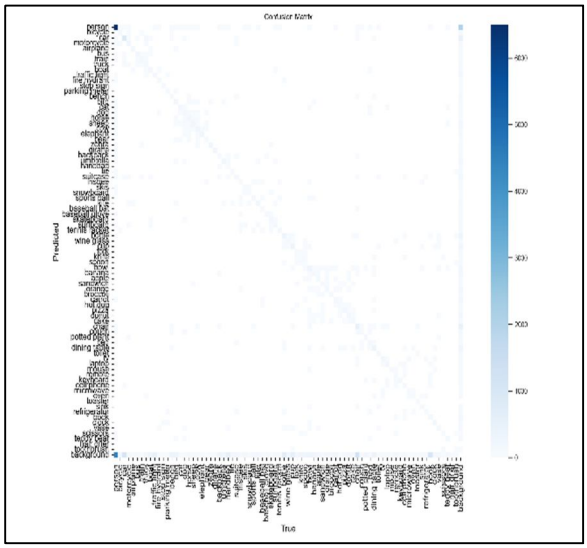


Fig.6 Confusion matrix

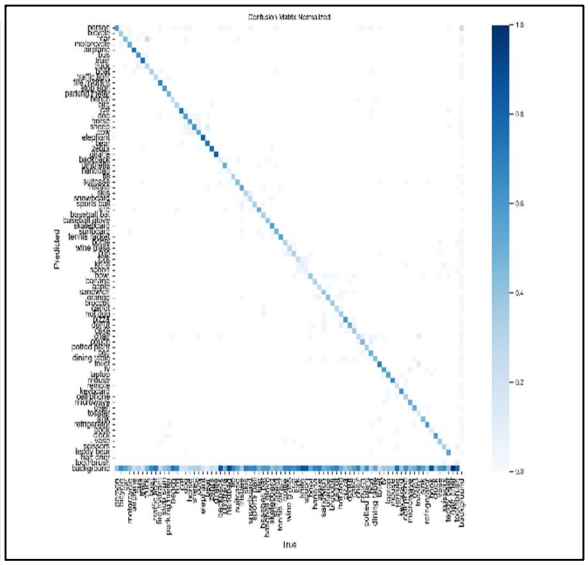


Fig.7 confusion matrix normalized

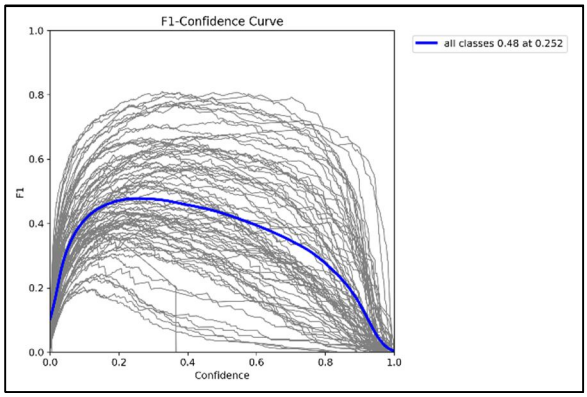


Fig.8 F1 Confidence curve

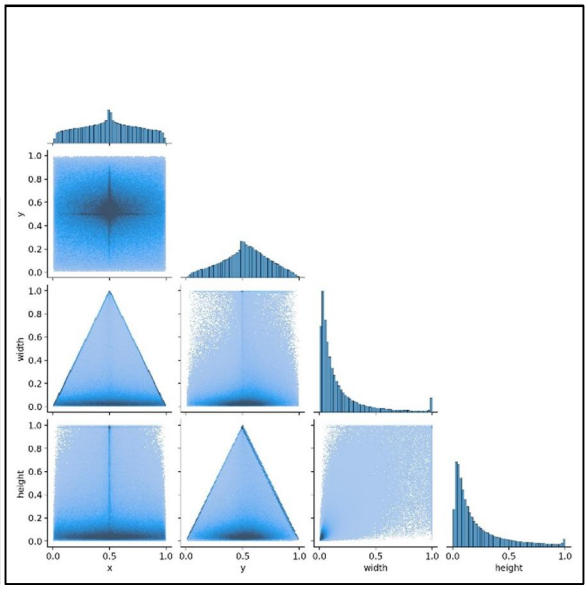


Fig.9 labels correlogram

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

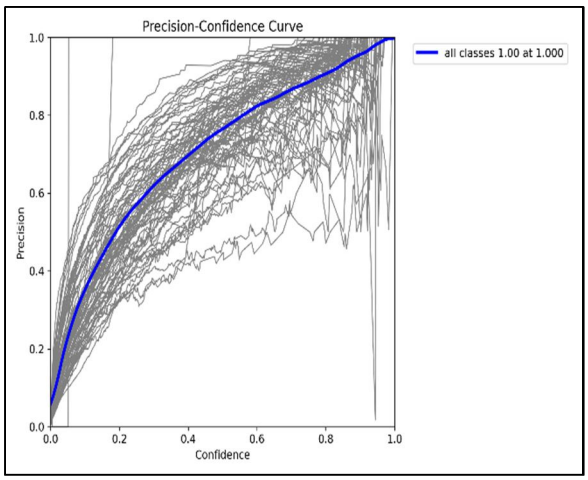


Fig.10 Precision-confidence curve

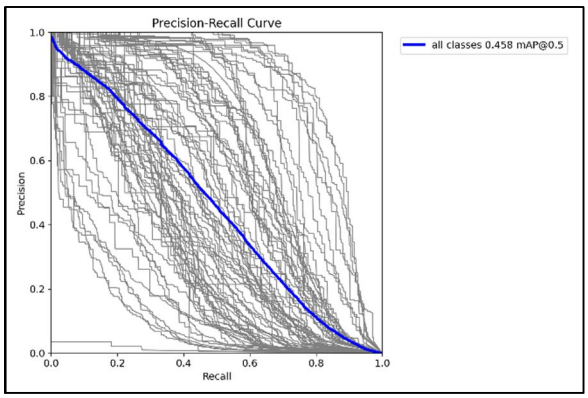


Fig.11 precision-recall curve

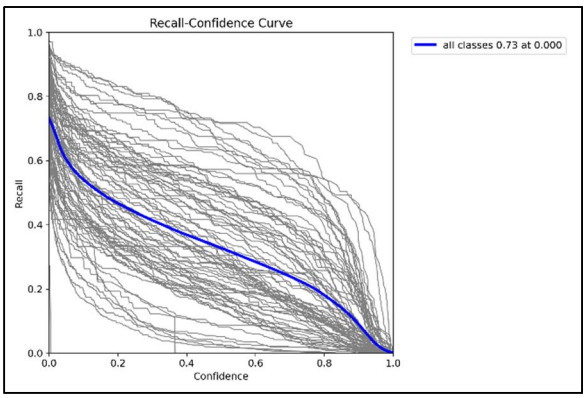


Fig.12 recall-confidence curve

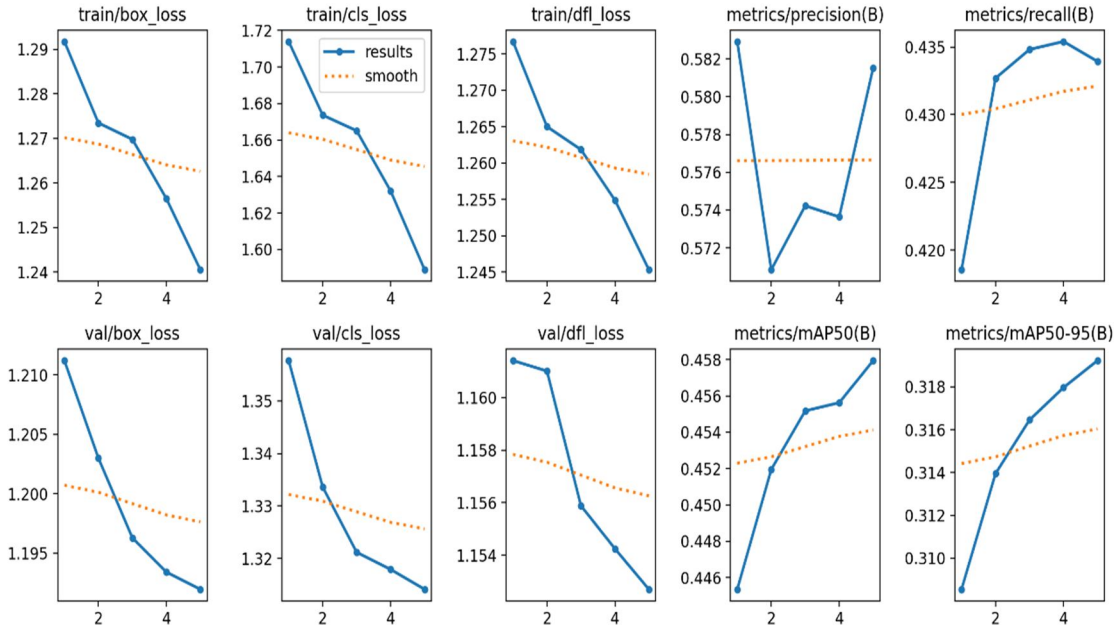


Fig.13 results

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

B. Accuracy and Evaluation Metrics

After 5 epochs of training, the following key metrics were observed on the validation set (val2017):

Table.1 Accuracy

Metric	**Value**
mAP@0.5	0.458
mAP@0.5:0.95	0.319
Precision	0.582
Recall	0.434

The mean Average Precision (mAP), which assesses both localization and classification performance, is a thorough measure of item detection accuracy. The average precision (AP) across all object classes is used to calculate it:

$$\text{mAP}=\frac{1}{N}\sum_{i=1}^{N} AP_i \quad (1)$$

Where:

- N is the total number of object classes,
- AP_i *i*s the Average Precision for the i^{th}class, derived from the area under the Precision-Recall curve for that class.

By using mAP computed at different IoU thresholds between 0.5 and 0.95 (in steps of 0.05), the COCO evaluation methodology offers a reliable evaluation of detection accuracy under different levels of overlap between the predicted and ground truth boxes.

C. Class-wise Performance Highlights

1) Highest performance:
  - Train: mAP@0.5 = 0.806
  - Airplane: mAP@0.5 = 0.777
  - Fire Hydrant: mAP@0.5 = 0.729

2) Lower performance:
  - Boat, Traffic Light, Bench, Toaster, Hair Drier had lower scores, especially for mAP@0.5:0.95.

D. Interpretation

1) The model is reasonably accurate for a lightweight version trained on CPU and only 5 epochs.

2) The yolov8n model has fewer parameters (~3.15M), making it fast but less accurate than larger models (s, m, l, x).

IV. FUTURE SCOPE

Future enhancements include enabling more epochs (≥100) for improved accuracy and training the model on a GPU (such as an RTX 3060) to save time. Performance can be further improved by hyperparameter adjustment (e.g., learning rate, optimizer, augmentation like mixup and mosaic). Investigating variations such as YOLOv8m or v8s could increase accuracy while using less resources. Class imbalance can be addressed with sophisticated data augmentation and methods such as focused loss. Models can be efficiently used on devices like as the Raspberry Pi or Jetson Nano by being translated to ONNX or compressed using INT8 quantization before distribution.

V. CONCLUSION

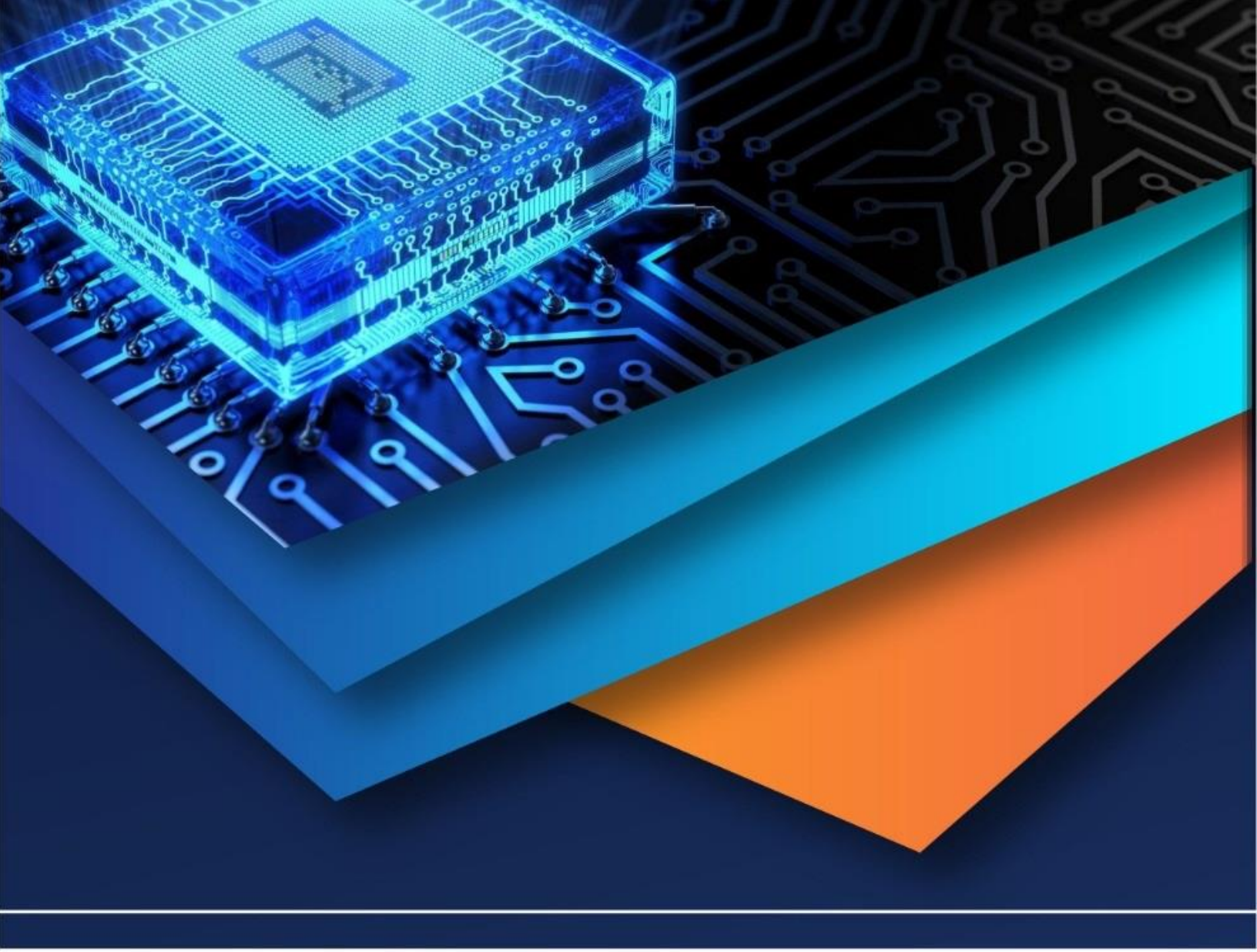
The experiment shows that YOLOv8n can be successfully trained on the COCO dataset on a CPU with constrained resources. After only 5 epochs, the model's mAP@0.5 of 45.8% and mAP@0.5:0.95 of 31.9% demonstrated its capacity to learn despite limitations. This creates a good baseline, even when accuracy is low because to the short training period and CPU limitation. In order to increase accuracy, it is advised that future studies use GPU training with more epochs and hyperparameter adjustment. Because of its modular design, pretrained weights, and speed, YOLOv8 is a great option for real-time applications such as smart surveillance systems on embedded devices or outdoor navigation for the blind and visually impaired.

©IJRASET: All Rights are Reserved

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[3] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2005, pp. 886–893, doi: 10.1109/CVPR.2005.177.
[4] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), vol. 1, pp. 511–518, 2001.
[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2014, pp. 580–587, doi: 10.1109/CVPR.2014.81.
[6] W. Liu et al., “SSD: Single shot multibox detector,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2016, pp. 21–37, doi: 10.1007/978-3-319-46448-0_2.
[7] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Advances in Neural Inf. Process. Syst. (NeurIPS), vol. 28, pp. 91–99, 2015.
[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, “SSD: Single shot multibox detector,” in Eur. Conf. Comput. Vis. (ECCV), pp. 21–37, 2016.
[9] S. N. Katkade, V. C. Bagal, R. R. Manza, and P. L. Yannawar, "Advances in Real-Time Object Detection and Information Retrieval: A Review," Artificial Intelligence and Applications, vol. 1, no. 3, pp. 123–128, Mar. 2023, doi: 10.47852/bonviewAIA3202456
[10] G. Jocher et al., “YOLO by Ultralytics,” GitHub Repository, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.
[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 779–788, 2016
[12] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” arXiv preprint arXiv:1804.02767, 2018
[13] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” arXiv preprint arXiv:2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934
[14] G. Jocher, “YOLOv5: A research project by Ultralytics,” 2020. [Online]. Available: https://github.com/ultralytics/yolov5
[15] M. Meituan, “YOLOv6: A single-stage object detection framework for industrial applications,” arXiv preprint arXiv:2209.02976, 2022.
[16] C. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” arXiv preprint arXiv:2207.02696, 2022. [Online]. Available: https://arxiv.org/abs/2207.02696
[17] Ultralytics, "YOLOv8 Documentation," 2023. [Online]. Available: https://docs.ultralytics.com/
[18] G. Jocher et al., “YOLOv8: The latest evolution in real-time object detection,” https://github.com/ultralytics/ultralytics, 2023 (accessed May 2025).
[19] S. N. Katkade, R. R. Manza, P. S. Shinde, and B. A. Balande, "Advances in Object Detection Algorithms Using Deep Learning," International Journal of Advanced Research in Computer and Communication Engineering, vol. 9, no. 5, pp. 1–5, May 2020, doi: 10.17148/IJARCCE.2020.951.
[20] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014. (Reference often cited for data augmentation including noise techniques
[21] R. Girshick, “Fast R-CNN,” Proc. IEEE Int. Conf. Comput. Vis. (ICCV), pp. 1440–1448, 2015.
[22] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” Proc. Int. Conf. Mach. Learn. (ICML), pp. 6105–6114, 2019.
[23] T.-Y. Lin, M. Maire, S. Belongie, et al., “Microsoft COCO: Common objects in context,” in Eur. Conf. Comput. Vis. (ECCV), 2014.
[24] L. Liu , H. Ouyang, X. Wang, et al., “Swin Transformer: Hierarchical vision transformer using shifted windows,” Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), pp. 10012–10022, 2021.
[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 770–778, 2016
[26] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” arXiv preprint arXiv:1711.05101, 2017.
[27] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.
[28] N. Narwal, A. Malviya, and S. K. Dwivedi, “YOLOv5: A performance evaluation,” in 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 1776–1781, doi: 10.1109/ICICCS51141.2021.9432097.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)