Performance Comparison between YOLOv11 and YOLOv8 on Traffic Sign Detection

Tom Solomon, Kenneth Bach, Shrikanth Shivakumar, Aaron De Castro

Electrical and Computer Engineering Department, California State Polytechnic University, Pomona
Pomona, CA, United States of America

tcsolomon@cpp.edu

sshivakumar@cpp.edu

kabach@cpp.edu

aodecastro@cpp.edu

***Abstract*— Traffic sign detection is a critical task in the development of intelligent transportation systems and autonomous vehicles, where real-time performance and high detection accuracy are essential. This paper presents a comparative study between two state-of-the-art object detection models, YOLOv8 and YOLOv11, for traffic sign recognition. The models were trained and evaluated on a robust dataset of annotated traffic sign images, leveraging cloud-based training with virtual GPUs and edge deployment on a Raspberry Pi 4B for real-world applicability.**

***Keywords*— Autonomous Vehicles, Deep Learning, Object Detection, Traffic Sign Detection, YOLO Models**

I. Introduction

Object detection is a method used by computers to recognize requested objects either with the use of visual devices such as cameras or from a set of already taken images. Research behind object detection has seen a boom in interest ever since the turn of the century, as seen in [1, Fig. 1].

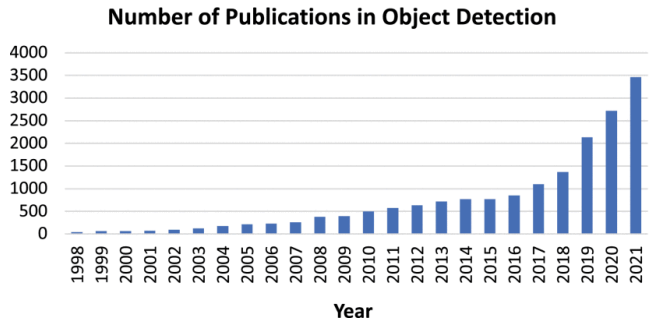

Fig. 1 Increasing number of publications in object detection from 1998 to 2021.

A. Hardware Used

The system is set up on the Raspberry Pi 4, a compact and powerful single-board computer that is used for edge computing applications. The RPi 4 uses its quad-core ARM processor that runs at 1.5 GHz to accomplish various tasks; it also has different RAM configurations to be used depending on the computational needs of the program. Since the system will be deploying traffic sign detection using YOLO models, the RPi's processing power and compatibility with the machine learning framework makes it a good fit for use in this project.

B. Software Used

The main focus of this comparison is YOLO, implemented using the PyTorch framework due to its flexibility and robust support for deep learning models. YOLOv8 and YOLOv11 were accessed through their official repositories, ensuring compatibility with the dataset and reproducibility of the results. Training was conducted using Google Colab Pro to leverage high-performance cloud-based GPUs, and model deployment was carried out using TensorRT to optimize inference times on the Raspberry Pi 4B. Additionally, Roboflow's platform was utilized for dataset preprocessing, augmentation, and formatting.

The Raspberry Pi operated with the latest Raspberry Pi OS (Debian-based), enabling compatibility with Python libraries required for edge deployment, such as OpenCV, NumPy, and PyTorch Lite. These tools

collectively supported the seamless execution of the YOLO models, from training to edge inference.

II. Methods

A. Dataset

The dataset utilized in this study was sourced from Roboflow and formatted for a variety of models such as YOLOv8, YOLOv11, and YOLO Darknet. It comprised a comprehensive collection of annotated traffic sign images, tailored for object detection tasks. Roboflow's platform ensured that the dataset was pre-processed and augmented for compatibility with modern computer vision models. This dataset played a pivotal role in training and testing the models for traffic sign recognition.

For the experiments, the dataset was partitioned into 91% for training, 6% for validation, and 3% for testing. The annotations were formatted in YOLOv8's specific structure.

The dataset's versatility made it suitable for evaluating the model's ability to generalize across varying lighting conditions, sign positions, and sign variations. Sample images from the dataset are presented in Fig. 1.

B. Training Process

The YOLOv8 model was trained using the Roboflow Google Colab notebook, leveraging the platform's integration with cloud-based virtual GPUs. Training deep learning models on local hardware can be time-intensive and resource-demanding. By utilizing virtual GPUs, faster training times were achieved, and hardware constraints were minimized. Training was conducted over 125 epochs with an image size of 640, with hyperparameters optimized based on recommendations provided by Roboflow's integration. The model's training progress was monitored using metrics such as precision, recall, and mAP50 (Mean Average Precision at threshold of 0.5).

1) Precision

Precision will measure the correctness of predictions, as high precision means fewer false positive predictions. This is defined with Eq. (1), where true positives are correct predictions and false positives are incorrect predictions.

$$Precision = \frac{True\ Positive}{(True\ Positive + False\ Positive)} \quad (1)$$

2) Recall

Recall will measure the wholeness of the captures, where a high recall value means the model captures most of the ground truth objects. This is defined with Eq. (2), where false negatives are cases that are incorrectly identified as negative.

$$Recall = \frac{True\ Positive}{(True\ Positive + False\ Negative)} \quad (2)$$

3) Mean Average Precision

The mAP50 is a metric for object detection accuracy, combining precision and recall values to reflect the model's accuracy within the standard of 50% IoU (Intersection over Union) overlap, as utilized in Eq. (3).

$$Average\ Precision(AP) = \alpha \int precision(r) dr \quad (3)$$

C. Deployment and Testing

After training, the model was initially deployed on a personal computer for testing and debugging. This phase allowed for thorough validation of the model's functionality and facilitated adjustments before deploying it to more constrained environments. Following successful validation, the model was deployed on a Raspberry Pi 4B device with the Raspberry Pi OS installed. This transition to an edge-computing platform ensured the feasibility of real-time traffic sign recognition in compact and portable setups.

D. Performance Assessment

To evaluate the network capabilities for real-time deployment, Perf was employed to measure computing performance under various conditions. In particular, the perf stat command was used to evaluate metrics such as CPU time, clock cycles, and instruction count.

E. YOLOv11 Comparison

In addition to YOLOv8, a parallel set of experiments was conducted using YOLOv11. The same dataset was reformatted for YOLOv11 compatibility, and the training and deployment processes were replicated. This comparison allowed for an assessment of model performance across different YOLO versions, highlighting differences in detection accuracy, speed, and computational efficiency.

III. Results

Upon training the YOLOv8 and YOLOv11 models using the same dataset of traffic signs, the elapsed times for training are provided in Table I and the results of certain significant classes are shown in Table II. 25 MPH, 35 MPH, 45 MPH, and STOP each had a larger sample size of images to train on, so these will be observed further.

TABLE I. Elapsed Time For Model Training

Model	Time (hours)
YOLOv8	1.690
YOLOv11	2.001

TABLE II. Best Model Comparison

Model	Class	Images	Precision	Recall	mAP50
YOLOv8	**All**	144	0.897	0.867	0.907
	25 MPH	20	0.940	0.78	0.946
	35 MPH	21	0.941	0.724	0.831
	45 MPH	15	0.918	0.789	0.930
	STOP	25	1	0.958	0.989
YOLOv11	**All**	144	0.826	0.869	0.899
	25 MPH	20	0.886	0.9	0.897
	35 MPH	21	0.878	0.653	0.929
	45 MPH	15	0.812	0.789	0.915
	STOP	25	1	0.929	0.976

As shown in Table II, the comparison between both models is nearly identical, regardless of an increase of 18.4% in the time taken to train the newer model. Fig. 2 and Fig. 3 are images provided by Roboflow representing security camera footage similar to ones to be used in automated vehicles. Both models are capable of correctly detecting traffic signs at a similar level of confidence. Since these are still images, this section of the test mostly focuses on the ability to detect a traffic sign at any moment it is clear to see from the camera rather than the ability to comprehend and predict a traffic sign when blurred or obstructed by going at high speeds.

Fig. 2. YOLOv8

Fig. 3. YOLOv11

Table III presents the performance evaluation of each model.

IV. Conclusion

This study presents a detailed performance comparison between YOLOv8 and YOLOv11 for traffic sign detection, with an emphasis on metrics such as precision, recall, and mAP50. The results

indicate that while both models exhibit comparable detection capabilities, YOLOv8 achieves slightly higher precision and mAP50 across most traffic sign classes, including STOP and speed limit signs. YOLOv8 also demonstrates faster training times, completing training approximately 18.4% more quickly than YOLOv11.

YOLOv11, however, achieves marginally higher recall for specific classes, suggesting a potential advantage in scenarios where comprehensive object detection is critical. Both models perform effectively in real-time detection when deployed on edge devices such as the Raspberry Pi 4B, demonstrating their suitability for traffic sign detection in automated vehicle systems.

Future work includes evaluating these models in dynamic environments, such as detecting signs in video frames captured at high vehicle speeds or under adverse weather conditions. These findings contribute to ongoing advancements in object detection, paving the way for the development of more robust and efficient intelligent systems.

References

[1] Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye, "Object Detection in 20 Years: A Survey," in Proceedings of the IEEE, vol. 111, no. 3, pp. 257-276, March 2023, doi: 10.1109/JPROC.2023.3238524.

TABLE III. Performance Comparisons

Model	**Performance**		
YOLOv8	Cycles	490,196,308,242	1.729 GHz
	Instructions	413,813,398,747	0.84 insn per cycle
	Cache-Misses	4,802,178,969	5.21% of all cache refs
	Cache-References	92,254,164,579	325.436 M/sec
	Branch-Misses	318,593,074	
	Task-Clock	283,478.49 msec	1.813 CPUs utilized
	Context-Switches	0	0.000 /sec
	Page-Faults	709,206	2.502 K/sec
	Time Elapsed	156.386003770 sec	
	User	275.720335000 sec	
	System	8.089143000 sec	
	FPS = 1/.9571s = 1.045 FPS		
	Cycles	539,804,227,494	1.702 GHz
	Instructions	448,732,835,022	0.83 insn per cycle
	Cache-Misses	4,984,938,173	5.01% of all cache refs
	Cache-References	99,595,990,618	314.072 M/sec

YOLOv11	Branch-Misses	345,711,518	
	Task-Clock	317,112.05 msec	1.881 CPUs utilized
	Context-Switches	0	0.000 /sec
	Page-Faults	1,427,568	4.502 K/sec
	Time Elapsed	168.586001219 sec	
	User	304.288101000 sec	
	System	13.142862000 sec	
	FPS = 1/1.0594s = 0.94429 FPS		