


Towards Real-time Vietnamese Traffic Sign Recognition on Embedded Systems

Phuong-Nam Tran 

Dept. of Computer Science and Engineering
Kyung Hee University
Yongin-si, Gyeonggi-do, Republic of Korea
tpnam0901@khu.ac.kr

Nhat Truong Pham 


Dept. of Integrative Biotechnology
Sungkyunkwan University
Suwon, Republic of Korea
truongpham96@skku.edu

Nam Van Hai Phan 

Dept. of Computing Fundamental
FPT University
Ho Chi Minh City, Vietnam
nampvhse182309@fpt.edu.vn

Duc Tai Phan 

Dept. of Computing Fundamental
FPT University
Ho Chi Minh City, Vietnam
phantaiduc2005@gmail.com

Cuong Tuan Nguyen 

Faculty of Engineering
Vietnamese-German University
Binh Duong, Vietnam
cuong.nt2@vgu.edu.vn

Duc Ngoc Minh Dang* 

Dept. of Computing Fundamental
FPT University
Ho Chi Minh City, Vietnam
ducnm2@fe.edu.vn

Abstract—AI development has brought many significant changes in various aspects of our daily lives in recent years. Integrating AI technology into various applications has revolutionized multiple domains, and one particularly vital area is traffic sign recognition, which significantly enhances driver safety. This paper presents an approach to traffic sign recognition specifically designed for the Jetson Nano 2GB device. By utilizing the YOLOv8 Nano model, the proposed approach achieves a remarkable frame rate of up to 32 frames per second (FPS). To optimize inference speed on Jetson with limited memory, the approach incorporates TensorRT and quantization techniques. In addition, this paper introduces a dataset called the Vietnamese Traffic Sign Detection Database 100 (VTSDB100). This dataset is an extension of the VTSDB46 dataset and encompasses a comprehensive collection of 100 different classes of traffic signs. These signs were captured in diverse locations within Ho Chi Minh City, Vietnam, providing a rich and diverse dataset for training and evaluating traffic sign recognition models. Extensive experiments and analyses were conducted using various object detection methods on the VTSDB100 dataset. The findings highlight the potential of deploying the proposed approach on resource-constrained devices and provide valuable insights for further research and development in AI-powered driver safety systems.

Index Terms—Traffic Sign Recognition; Object Detection; Quantization; Vietnamese Traffic Sign dataset; Deep learning

I. INTRODUCTION

Traffic sign recognition (TSR) has been a well-studied topic. However, its recent implementation in the Internet of Things (IoT) domain faces limitations due to hardware constraints, making it challenging to deploy high-performance deep learning models on IoT devices. Furthermore, achieving high frame rates on these devices poses additional difficulties, especially when memory and hardware resources are limited. As a result, deep learning approaches become costly to implement on such devices. To tackle this issue, we leverage various techniques

and frameworks to reduce the model size and memory requirements for running deep learning models efficiently.

Furthermore, this paper conducts experiments using recent deep-learning methods to assess their performance on the VTSDB100 dataset. Faster R-CNN [1] is selected for two-stage object detection due to its high accuracy. As for one-stage object detection, popular models such as You Only Look Once (YOLO) [2] and its variants, YOLOX [3] and RetinaNet [4], are utilized to observe their performance on our dataset. These experiments establish a baseline performance on VTSDB100, serving as a valuable reference for future research.

Our primary contributions in this paper cover three main aspects:

- 1) We introduce a new dataset, Vietnamese Traffic Sign Detection Database 100 (VTSDB100), which extends the previous VTSDB46 dataset, providing a more comprehensive collection of Vietnamese traffic signs for object recognition research.
- 2) We conduct a baseline performance evaluation on the VTSDB100 dataset, utilizing various recent deep-learning models. This serves as a valuable reference for future research in the field.
- 3) We successfully deploy deep learning models on a resource-constrained Jetson Nano device with 2GB RAM, achieving high-speed performance through quantization techniques and TensorRT.

The remainder of this paper is organized as follows. In Section II, we provide an overview of existing research on TSR. Then, in Section III, we explain the deep learning architecture used in our experiment and how we compressed the model to make it suitable for running on IoT devices. Next, we present the results of our experiments on different deep learning architectures using the VTSDB100 dataset and IoT devices in Section IV. Finally, we wrap up our study in

* Corresponding author: Duc Ngoc Minh Dang (ducnm2@fe.edu.vn)

Section V by selecting the best model with the most effective performance. We also discuss its potential applications and future developments in this field.

II. RELATED WORK

Various techniques have been employed to address the challenge of traffic sign recognition, with deep learning methods being the most popular and practical approach. As artificial intelligence continues to advance, object detection techniques have evolved to become faster and more accurate. One notable technique in object detection is the introduction of two-stage object detection, proposed by Faster R-CNN [1]. This approach allows deep learning models to achieve real-time object detection capabilities. However, researchers have since proposed a simplified alternative known as one-stage object detection. One of the most popular methods in this architecture is YOLO [2]. YOLO takes a single holistic approach to object detection, directly predicting bounding boxes and class probabilities in a single pass. This design allows for faster inference speed compared to two-stage methods. Building upon the YOLO architecture, subsequent variants such as YOLOv8 [5], YOLOv9 [6], and the most recent iteration, YOLOv10 [7], have been introduced. These iterations aim to improve further the performance and accuracy of the original YOLO approach. Additionally, other object detection architectures have emerged based on the YOLO framework, including YOLOX [3] and YOLO-NAS [8]. These architectures leverage the strengths of YOLO while introducing novel enhancements and optimizations. Overall, the development and evolution of these object detection architectures, particularly the YOLO family, have significantly contributed to advancing the field and expanding the potential applications of real-time object detection, particularly in the realm of IoT devices.

Traffic sign detection has been a popular topic in recent years, and various methods and datasets have been introduced to bring the application of this field into real life. German Traffic Sign Detection Benchmark dataset (GTSDB) [9] is one of the most popular datasets in Germany that is created for traffic sign object detection. GTSDB consists of 900 images and annotations that provide information about the traffic signs' locations. However, this dataset doesn't specify the type of traffic sign present in each image and is mainly designed for benchmarking purposes rather than practical use in real-world scenarios. Another popular dataset is the Tsinghua-Tencent 100K [10] dataset, considered the most suitable for meeting real-time application requirements in China. This dataset consists of 90,000 Tencent Street View panoramas, with around 30,000 instances of traffic signs. It captures images under different lighting and weather conditions, reflecting real-world scenarios. The Tsinghua-Tencent 100K dataset offers a high-quality collection that can be effectively utilized for traffic sign applications in China. However, it is essential to note that this dataset is captured in a panoramic format, which may result in reduced performance when applied to regular cameras. In the case of Vietnam, using existing datasets such as the Tsinghua-Tencent 100K may result in lower performance due to the

discrepancies and potential absence of Vietnamese traffic signs in those datasets. This is because the traffic sign designs and characteristics can vary between countries. VTSD46 [11] and GPVNS [12] are the two latest traffic sign datasets available in Vietnam. While GPVNS only consists of 1088 samples across 29 different classes, the VTSD46 dataset presents a more extensive collection, comprising 80,000 samples categorized into 46 distinct classes. This larger dataset provides an ideal resource for the development of applications. The VTSD46 was captured in numerous locations throughout Ho Chi Minh City, Vietnam. With its quality recording and accurate labeling, VTSD46 offers a high-quality dataset tailored to Vietnamese traffic signs. This dataset is suitable for real-time traffic sign recognition applications in Vietnam. However, VTSD46 still has limitations, as it does not encompass all the traffic sign classes in Vietnam. Therefore, further efforts are needed to create a more comprehensive dataset for complete traffic sign recognition applications in the country.

III. METHODOLOGY

A. Vietnamese Traffic Sign Detection Database 100 - VTSD100

The VTSD46 [11] dataset is a collection of traffic sign data recorded in 12 districts of Ho Chi Minh City, Vietnam. It focuses on traffic signs in urban environments, considering variations in lighting, background, and weather conditions. However, this dataset includes only 46 types of traffic signs from the four main groups of Vietnamese traffic signs. To expand the dataset, we created VTSD100, which contains 100 classes covering the most common Vietnamese traffic signs.

The VTSD100 dataset contains more than 100,000 samples created by augmenting over 35,000 original samples. These original samples were captured using a GoPro camera. To annotate the dataset, we utilize a tool called Label Studio[13]. This tool provides a user-friendly interface and efficient functionality for annotation purposes. The annotation process involves manually labeling the 35,000 samples and converting the dataset into a YOLO format commonly used for object detection tasks. During the annotation process, we focus on labeling only those traffic signs that can be easily identified by human eyes. If a traffic sign causes confusion or its content is not fully visible and readable, we do not label it. For instance, if a traffic sign displays only the number "5" instead of the complete "50" indicating a speed limit of 50, we do not include it in the annotations.

The VTSD100 dataset is split into training, validation, and testing sets. Before dividing the VTSD100 dataset, we performed data augmentation using the Albumentations [14] library, which provides various augmentation methods to augment the dataset. These methods include random cropping with a probability of 1.0, random adjustments to brightness and contrast with a probability of 0.5, random gamma adjustments with a probability of 0.4, random rotation up to 20 degrees with a probability of 0.4, random rain effects with a probability of 0.2, random fog effects with a probability of 0.2, and

random contrast-limited adaptive histogram equalization with a probability of 0.4. We randomly selected 100 samples per class from the VTSDDB100 dataset to create the testing set. This approach ensured that the testing set contained a diverse representation of samples from each class, providing an independent evaluation of the model’s performance. Subsequently, we divided the remaining samples into two subsets: the training and validation sets. For the validation set, we continued randomly selecting 100 samples per class from the remaining dataset. The random selection process considered the availability of samples for each class. Therefore, the number of samples in the validation and testing sets may be smaller than the target of 10,000 samples. Table I illustrates the number of training, validation, and test subsets samples in the VTSDDB100 dataset.

TABLE I
THE DISTRIBUTION OF TRAINING, VALIDATION, AND TESTING SETS IN
THE VTSDDB100 DATASET.

Subset	Number of samples
Training	77,499
Validation	7,496
Testing	7,519

B. Proposed methods

1) *Object detection*: In this work, we evaluated various object detection models using our new dataset. Specifically, we examined three recent methods from the YOLO (You Only Look Once) family, namely YOLOv8 [5], YOLOv9 [6], and YOLOv10 [7]. These methods are known for their one-stage object detection approach, offering a balance between speed and performance with minimal parameters. The feature pyramid network [15] employed in the YOLO family contributes to its high performance and efficiency. To accommodate the use of IoT devices, we focused on training and experimenting with the smallest and fastest models within each YOLO method. However, if larger architectures demonstrated superior performance, they were also considered in our work. In addition to the YOLO family, we also experimented with YOLOX Nano [3], which introduces a novel approach to object detection. Unlike other YOLO methods, YOLOX Nano predicts an object’s location without relying on anchor boxes. It employs a decoupled head architecture, separating class predictions from bounding box regression. By combining anchor-free detection, the decoupled head, and advanced techniques, YOLOX Nano achieves improved object detection performance, making it suitable for real-time tasks.

In our two-stage object detection experiments, we utilized the Faster R-CNN [1] architecture with the MobileNetV3 [16] backbone. Faster R-CNN is one of the earliest and most widely used methods known for its high performance in object detection tasks. The Faster R-CNN architecture consists of two main components. First, a convolutional neural network (CNN) is employed to extract features from the input image. These features capture hierarchical representations of the image, enabling the network to understand the visual content.

The second component is the region proposal network (RPN), which operates on the extracted features. The RPN predicts regions of interest or anchor boxes that likely contain objects. These regions are potential candidates for object detection. Once the regions of interest are identified, a fully connected layer is utilized to predict the class labels and refine the bounding box coordinates for the detected objects. The strength of Faster R-CNN lies in its two-stage network architecture. By leveraging the feature extraction capabilities of CNNs and the region proposal mechanism, Faster R-CNN achieves high accuracy in object detection. However, due to the two-stage process, the speed of Faster R-CNN is slower compared to one-stage architectures.

2) *IoT device selection*: Deploying a deep learning model on an IoT device is crucial for bringing the application into real-life scenarios. In such a system, it is essential to consider the constraints of power supply, hardware resources, and cost-effectiveness. To address these requirements, we chose the Jetson Nano 2GB as the baseline platform for our application. The Jetson Nano 2GB is a cost-effective IoT device with limited RAM but offers acceptable performance for deep learning tasks. Despite having only 2GB of RAM, the presence of CUDA cores on the Jetson Nano 2GB enables efficient computation for deep learning models. CUDA cores are parallel processors specifically designed to accelerate deep learning workloads, providing a significant boost in performance compared to traditional CPUs. By utilizing the Jetson Nano 2GB, we can deploy deep learning model in cars, where low power consumption and limited hardware resources are critical.

On the other hand, the Jetson Orin Nano 8GB is a more powerful option than the Jetson Nano. It has a GPU that contains eight times the number of CUDA cores compared to the Jetson Nano. Hence, it can handle more complex deep-learning calculations and models. The Jetson Orin also has better RAM, allowing larger models to be loaded and run in real time. However, there are some drawbacks to using the Jetson Orin. Firstly, it is more expensive than the Jetson Nano, so the cost of implementing an application using the Jetson Orin would be higher. Additionally, the power supply becomes a challenge for Jetson Orin. While the Jetson Nano only utilizes a 5V power supply, the Jetson Orin needs a minimum of 15V power supply to function correctly. Therefore, additional power considerations and costs need to be taken into account when deploying in real-life applications.

3) *Model compression*: To enhance performance with limited 2GB RAM, it is important to select the appropriate framework and deep learning model. In this research, we utilize the TensorRT framework, specifically designed for NVIDIA Jetson devices. TensorRT is a software development kit that enables efficient deep learning inference on NVIDIA GPUs. It is optimized for the Jetson Nano, enabling high-performance deep-learning models. To integrate our model with TensorRT, we first convert our PyTorch model to the ONNX format to ensure compatibility. Then, we apply a quantization process to the ONNX model. Quantization involves converting the

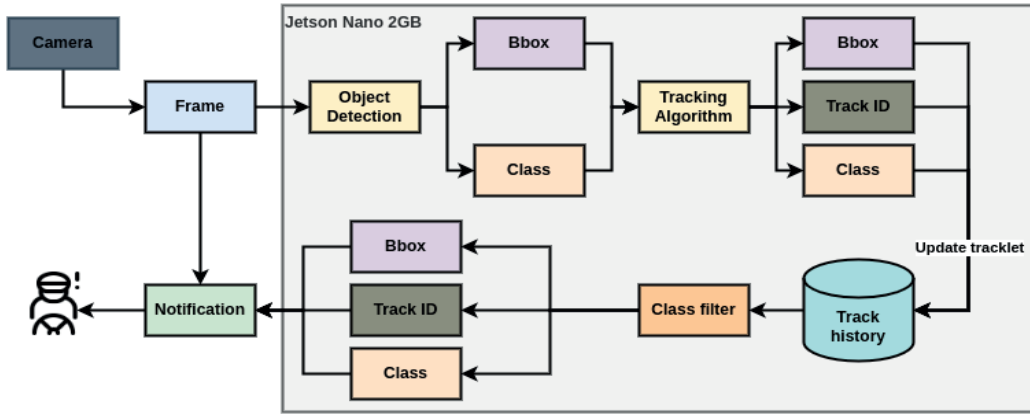


Fig. 1. Real-Time Traffic Sign Recognition Workflow on Jetson Nano 2GB using Object Detection with Integrated Tracking Algorithm.

model’s data type from float32 to int8, reducing the size of model parameters, and improving model speed. However, it is essential to note that the quantization process has some limitations. One limitation is that the model’s accuracy may decrease due to the limited number of available data types with int8. Finally, we convert the model from ONNX to TensorRT format using the conversion provided by NVIDIA. The final conversion takes place on Jetson devices to ensure TensorRT package compatibility, while the other conversions and quantization processes are performed on the server.

4) *Workflow*: The design for the TSR system is depicted in Fig. 1. The process starts with the Jetson Nano receiving an image from the camera in car. The Jetson Nano then uses deep learning to analyze the image and predict the location and type of the traffic sign. A simple tracking method is used to track the traffic signs, which calculates the distance between the predicted bounding box’s center and the historical center of the bounding box. The pair with the minimum distance is selected and compared against a threshold value. A new ID is assigned to the traffic sign if the distance exceeds the threshold. The traffic sign retains its previous ID if the distance is below the threshold.

Once the tracking ID is obtained, the relevant information, such as the bounding box coordinates, class label, and tracking ID, is added to a tracklet. A tracklet is a database element that maps the track ID to a list of bounding boxes and corresponding class labels. Each entry in the tracklet represents the location and type of the traffic sign at a specific frame in the past. The length of the tracklet can vary depending on the implementation, typically 10 frames in our experiments.

Before sending notifications to the driver, a class filter algorithm is applied to minimize false predictions. The complete message is sent to the driver only if the number of elements in the tracklet is exceeds a certain threshold. In such cases, the traffic sign type assigned to the message is determined by the most frequently occurring class within the tracklet. This approach helps reduce false predictions caused by noise and other factors.

IV. EXPERIMENTS

A. Model setup

We employ the default hyperparameters described in their works to optimize all models’ performance. We change the batch size to 32 and train these models for 50 epochs. During training, we employ an early stopping strategy where the models are evaluated on the validation dataset after each epoch, and the weights are saved based on the best results metric value obtained on the validation dataset. We utilize pre-trained models from the COCO dataset for all methods to initialize the model weights. This approach enables the models to leverage the knowledge learned from a large and diverse dataset, which can help improve their performance and reduce the training cost.

All the models are experimented on four different environments: Nvidia K80, Intel Core i9 12700K, Jetson Orin Nano, and Jetson Nano. The Intel Core i9 12700K and Nvidia K80 are components of a workstation. The Intel Core i9 12700K is the CPU, and the Nvidia K80 is the GPU. This workstation has 64GB of RAM and 1TB of SSD memory. The Jetson Orin Nano is an IoT device developed by NVIDIA that features 1024 CUDA cores and 8GB of RAM. The Jetson Nano is another NVIDIA IoT device but with fewer resources. It has only 128 CUDA cores and 2GB of RAM. All these environments are set up on Linux systems. The Jetson Nano uses Ubuntu 16.04, the Jetson Orin utilizes Ubuntu 20.04, and the workstation is set up with Debian Bookworm.

B. Model performance

1) *Metrics*: We evaluate model performance and speed using two common metrics: mean average precision (mAP) and multiply-accumulate operations (MACs). mAP, a widely used metric in computer vision for assessing object detection algorithms or models, calculates the average precision across all classes, providing insights into the accuracy of predictions for different object classes. In our experiment, we employ the COCO API to evaluate model performance, resulting in a mAP@50 (intersection over union threshold of 0.5) and a

TABLE II
PERFORMANCE COMPARISON OF VARIOUS DEEP LEARNING MODELS ON THE VTSD100 DATASET

Method	Variant	Training (%)		Validation (%)		Testing (%)	
		mAP@50	mAP@50:95	mAP@50	mAP@50:95	mAP@50	mAP@50:95
YOLOv10	Nano	0.974	0.809	0.969	0.796	0.979	0.808
YOLOv8	Nano	0.973	0.810	0.969	0.800	0.979	0.810
YOLOv8	Small	0.979	0.834	0.974	0.819	0.984	0.831
YOLOv8	Medium	0.980	0.856	0.975	0.838	0.986	0.851
YOLOX	Nano	0.954	0.728	0.942	0.713	0.944	0.708
RetinaNet	Resnet50	0.985	0.840	0.959	0.777	0.979	0.815
Faster R-CNN	MobileNetV3-Large	0.975	0.823	0.954	0.764	0.970	0.793

mAP@50:95 (intersection over union thresholds ranging from 0.5 to 0.95 with a step size of 0.05).

MACs are used to measure the complexity and speed of the model. They count the number of multiply and add operations performed by the deep learning model while ignoring other operations. Another metric for evaluating model speed is floating-point operations per second (FLOPs), which counts the number of floating-point operations in a deep learning model per second. Both MACs and FLOPs are metrics that indicate the speed or efficiency of a deep learning model based on the number of operations it performs. However, the values obtained for MACs and FLOPs may vary for the same architecture depending on how the researcher implements them. To address this variation, we use a Python library called Thop. This library considers only the number of multiplications in the deep learning model and ignores all the other operations, making the result value more general. In addition, the MACs value obtained using this library is approximately half of the FLOPs value using the other library.

2) *Results and analysis:* Table II illustrates the performance of different deep learning models on the VTSD100 dataset. We aimed to find a suitable model for IoT devices with high performance and low latency. Therefore, we focused on using small models for training and evaluation while including some large models for performance comparison. The table shows that RetinaNet [4] with ResNet50 [17] backbone achieved excellent results during training with mAP@50 of 0.985 and mAP@50:95 of 0.840. However, its performance significantly decreased on both the validation and testing sets due to overfitting on the training set. On the other hand, YOLOv8 Nano [5], which has a smaller backbone, performed better than RetinaNet on the validation and testing datasets. The YOLOv8 Nano model has a compact architecture and incorporates a feature pyramid network, which allows it to capture better features while maintaining high-speed performance and minimal parameters. This makes YOLOv8 Nano a more suitable choice for IoT devices, where resource constraints are essential to consider. Interestingly, the performance of YOLOv10 Nano [7], a newer version of the YOLO series, did not surpass that of YOLOv8 Nano. YOLOv10 Nano showed a slight decrease in performance when evaluated on our validation and testing sets using the mAP@50:95 metric.

When comparing the MACs and speed of these models, YOLOv8 Nano [5] stands out as the fastest among them, except for YOLOX Nano [3], as shown in Table III. The

average inference speed in the table is calculated by the average inference time of all samples in the validation dataset. Despite having fewer parameters, YOLOv10 Nano [7] still takes longer for inference than YOLOv8 Nano. YOLOv8 Nano has over 0.5 million parameters but smaller MACs, resulting in faster inference times. YOLOv8 Nano’s inference time is up to 2 ms faster than YOLOv10 Nano on the Tesla K80 and 16 ms faster on the Intel i9-12900K under the same testing conditions, respectively. However, the smaller number of parameters in YOLOv10 Nano still makes it worth further research, considering it achieves approximately the same mAP as YOLOv8 Nano.

On the other hand, YOLOX Nano demonstrates high speed and low memory requirements due to its anchor-free algorithm. This makes YOLOX Nano suitable for deployment on IoT devices with lower hardware capabilities than the Jetson Nano 2G. However, YOLOX Nano’s performance is worse than that of the other models, and it further decreases when techniques like quantization are applied to reduce the size and latency of the models. While achieving high speed on different hardware, when implemented on the Jetson Nano using the TensorRT framework, the YOLOX Nano appears slower than the YOLOv8 Nano.

Our experiments with the dataset and various hardware led us to select YOLOv8 Nano as the model for conversion from PyTorch to TensorRT, specifically for running on the Jetson Nano 2GB in real-life applications. Deploying real-time applications on the Jetson Nano presents challenges due to its limited RAM and CUDA cores. Running a model with an input size of 640x640 consumes a significant amount of RAM, leading to inference failures or significantly slower performance. Additionally, the performance is affected by the input from the camera, especially with high-resolution cameras that require more RAM for resizing in the pre-process and post-process methods. To address these issues, we modified the model by changing the input size from 640x640 to 480x480 for inference. We have also performed a fine-tuning process to adapt to this input size on the trained model. With these adjustments, we achieved a frame rate of up to 32 FPS for 720p videos on the Jetson Nano 2GB. However, the FPS may decrease with higher-resolution cameras, impacting the pre-processing and post-processing methods. For optimal performance on this device, a 480p camera resolution is recommended, though lower resolutions sacrifice detection range due to information loss in the image.

TABLE III
COMPARISON OF INFERENCE SPEED ACROSS DIFFERENT ARCHITECTURES

Method	Variant	Params (M)	MACs (G)	Average Inference Speed (ms)			
				Tesla K80	Intel i9-12900K	Jetson Orin	Jetson Nano
YOLOv10	Nano	2.839	4.527	12.785	57.568	39.912	131.350
YOLOv8	Nano	3.346	4.862	10.638	41.999	35.262	82.767
YOLOv8	Small	11.174	14.430	21.088	91.008	40.785	111.717
YOLOv8	Medium	25.914	39.692	48.758	199.242	44.255	335.377
YOLOX	Nano	0.916	1.328	4.273	15.788	29.959	83.744
RetinaNet	Resnet50	34.255	154.598	352.348	3089.883	N/A	N/A
Faster R-CNN	MobileNetV3-Large	19.457	4.726	45.983	244.619	N/A	N/A

V. CONCLUSIONS

In this study, we introduced VTSDB100, the Vietnamese Traffic Sign Database 100. This database consists of traffic sign data collected in Ho Chi Minh City, Vietnam. The purpose of VTSDB100 is to facilitate the application of self-driving cars in Vietnam. Our experiments have demonstrated the feasibility of running deep-learning models on IoT devices. This implementation involves techniques including quantization and selecting suitable inputs for IoT devices. Additionally, we extensively evaluated various deep-learning models for object detection. This evaluation provides valuable insights for choosing the most effective method for production purposes.

In future work, we plan to integrate the location of traffic signs on Google Maps into IoT devices. This integration will allow for the real-time updating of traffic signs, enabling drivers to receive notifications about upcoming signs before they are visible to the camera. Additionally, we will explore labeling methods such as active learning to reduce the human effort required.

REFERENCES

- [1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.
- [3] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.
- [4] R. Del Prete, M. D. Graziano, and A. Renga, "RetinaNet: A deep learning architecture to achieve a robust wake detector in sar images," in *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*, 2021, pp. 171–176.
- [5] R. Varghese and S. M., "YOLOv8: A novel object detection algorithm with enhanced performance and robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 2024, pp. 1–6.
- [6] S. Aharon, Louis-Dupont, Ofri Masad, K. Yurkova, Lotem Fridman, Lkdci, E. Khvedchenya, R. Rubin, N. Bagrov, B. Tymchenko, T. Keren, A. Zhilko, and Eran-Deci, "Super-Gradients," 2021. [Online]. Available: <https://zenodo.org/record/7789328>
- [7] C.-Y. Wang, I.-H. Yeh, and H. Liao, "YOLOv9: Learning what you want to learn using programmable gradient information," *ArXiv*, vol. abs/2402.13616, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:267770251>
- [8] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "YOLOv10: Real-time end-to-end object detection," *ArXiv*, vol. abs/2405.14458, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:269983404>
- [9] A. F. De Souza, C. Fontana, F. Mutz, T. A. de Oliveira, M. Berger, A. Forechi, J. de Oliveira Neto, E. de Aguiar, and C. Badue, "Traffic sign detection with VG-RAM weightless neural networks," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–9.
- [10] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, "Traffic-sign detection and classification in the wild," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2110–2118.
- [11] D. T. Nguyen, M. K. Phan, P.-N. Tran, and D. N. M. Dang, "Vietnamese traffic sign recognition using deep learning," in *Proceedings of the 2024 International Conference on Intelligent Information Technology*. New York, NY, USA: Association for Computing Machinery, 2024.
- [12] B. Dang Hai, H. D. Nguyen, T. N. Vo, P.-N. Tran, C. T. Nguyen, and D. N. M. Dang, "Performance comparison in traffic sign recognition using deep learning," in *Industrial Networks and Intelligent Systems*, N.-S. Vo, D.-B. Ha, and H. Jung, Eds. Cham: Springer Nature Switzerland, 2024, pp. 122–138.
- [13] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2022, open source software available from <https://github.com/heartexlabs/label-studio>. [Online]. Available: <https://github.com/heartexlabs/label-studio>
- [14] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [Online]. Available: <https://www.mdpi.com/2078-2489/11/2/125>
- [15] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944.
- [16] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.
- [17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.