Performance Evaluation of Edge Computing Object Detection Models for Maritime Surveillance on a Raspberry Pi

Hyago V. L. B. Silva, Felipe A. P. de Figueiredo, and Samuel B. Mafra
Instituto Nacional de Telecomunicações - Inatel
Santa Rita do Sapucaí, Brazil
hyago.silva@mtel.inatel.br, [felipe.figueiredo, samuelbmafra]@inatel.br

***Abstract*—The exponential growth of maritime traffic has introduced significant business opportunities while also posing unprecedented challenges for surveillance. Human-based surveillance is increasingly inadequate for managing the volume and complexity of maritime activities, necessitating innovative solutions. Therefore, this study presents a comparative analysis of five prominent object detection models applied to maritime surveillance: FOMO, MobileNetV2 SSD, YOLOv5, YOLOv8, and YOLOv10. By evaluating their performance metrics in maritime environments, this research identifies the most suitable model for specific applications. For instance, YOLOv8n and YOLOv10n excel in accuracy, making them ideal when precision is paramount, irrespective of inference time and resource consumption. In contrast, FOMO is preferable when resource efficiency and shorter inference times are critical. YOLOv5n offers a balanced approach between accuracy and inference speed, suitable for scenarios requiring both performance and efficiency. This study aims to identify the most appropriate object detection model for an IoT-based maritime surveillance application focused on vessel detection, thereby enhancing operational effectiveness and decision-making. Additionally, this study proposes a novel dataset with 4,998 annotated images of 10 different ship classes, further contributing to the field.**

***Index Terms*—Computer Vision, Object Detection, Internet of Things, Artificial Intelligence, Maritime-Surveillance.**

I. Introduction

With the surge in maritime activity, trade, and passenger transportation, the demand for autonomous monitoring systems has increased significantly [1]. This necessity arises from the limitations of manual monitoring, which relies on human resources and is prone to errors. Traditionally, maritime surveillance ground stations have been equipped with sensors such as sonar, radar, and cameras. Leveraging cameras in these locations presents an opportunity to develop intelligent applications using Computer Vision (CV) techniques. Cameras provide a cost-effective solution for low-energy applications, especially when combined with efficient, fast, and robust detection techniques on embedded systems with CV capabilities [2], [3], [4].

CV-based embedded applications must be scalable to integrate into different coastal areas, necessitating the adoption of the Internet of Things (IoT). The IoT paradigm connects physical devices embedded with cameras and other components to the Internet, enabling seamless data collection, exchange, and processing. This technology offers several advantages in the maritime context, including real-time surveillance, remote control, and data-driven decision-making across numerous locations simultaneously [5].

Edge Machine Learning (EdgeML), an emerging field at the intersection of edge and embedded computing and machine learning, offers promising solutions for cost and resource-efficient object detection (OD) and tracking (OT) in maritime surveillance tasks [6]. State-of-the-art OD and OT models can be employed to identify, classify, and track a variety of objects on the water's surface, including ships, smaller vessels, and even individual personnel [7]. Furthermore, they can be used to distinguish between lawful operations and illicit activities, such as smuggling, piracy, or unauthorized fishing, enabling authorities to respond swiftly and effectively [8].

Integrating OD and OT models into existing maritime surveillance systems can significantly enhance their capabilities, ensuring comprehensive coverage, rapid alerts, reduced human workload, and improved decision-making for human operators [9]. However, such models often exhibit high computational complexity, necessitating expensive hardware for efficient operation, which can hinder their deployment on resource-constrained IoT devices. Fortunately, techniques such as quantization can be employed to adapt these models for use on devices with limited resources. These methods reduce computational demands while maintaining the model's accuracy at an acceptable level [10].

Therefore, this study addresses the challenge of OD on resource-limited IoT devices for maritime surveillance. The study assesses and compares the performance of five OD models: Faster Objects More Objects (FOMO), MobileNetV2 Single Shot Detector Feature Pyramid Network Lite (MobileNetV2 SSD FPN Lite), You Only Look Once v5 Nano (YOLOv5n), YOLOv8n, and YOLOv10n in terms of detection precision, frame processing speed, and resource usage. These models and their quantized versions are embedded and tested on a Raspberry Pi 4 with a simple RGB camera. The aim is to identify the most effective model for specific requirements, whether prioritizing processing speed, detection precision, or a balanced approach between speed, precision, and resource consumption. Additionally, this work also contributes a new dataset with 4998 images of ten different ship classes to the literature.

The remainder of the article is structured as follows. Section II presents and discusses related works found in the literature. Section III describes the methodology employed in this study. Section IV presents and analyzes the results obtained from a series of experiments. Finally, Section V concludes the study with key insights and outlines potential directions for future research.

II. Related Works

In [11], the authors utilize optimized versions of YOLOv4 for marine OD, covering a range of targets including ships, submarines, marine animals, and infrastructure, and implement these models on three-edge devices. Despite being very expensive, the Nvidia Jetson Xavier AGX showed the best performance with an inference speed of 90 frames per second (FPS) and a limited degradation of 2.4% in average accuracy. On the other hand, the more economical but still costly Kria KV260 AI Vision Kit performed less well but with significantly lower power consumption. The Movidius Myriad X Vision Process Unit (VPU) efficiently balanced performance and energy efficiency. The work employed a dataset comprising 10,000 annotated images split into 15 classes, which included not only various types of ships but also other objects commonly found in marine environments.

In [12], YOLOv4 achieved a high mean Average Precision (mAP) of 93.55% and a fast detection speed of 43 FPS in detecting ship targets, outperforming other algorithms such as Faster R-CNN, SSD, and YOLOv3. The customized data set, consisting of 4,000 annotated images from nine categories of ships, was used to train the model. Strategies such as K-means clustering for optimizing prior bins, modifying the model structure, and using mixups were crucial to improving the algorithm's performance and increasing prediction accuracy and overall robustness. The proposed model was trained and tested on the cloud and achieved good results. In conclusion, the YOLOv4 algorithm has shown remarkable results in ship target detection, with high accuracy, real-time performance, and the ability to detect various categories of ships effectively.

In [13], the authors present an approach to ship detection that combines YOLOv8 with MobileViTSF and GSConv to enhance accuracy and efficiency, particularly for detecting small targets in complex environments. By integrating YOLOv8 with MobileViTSF and replacing traditional convolution blocks with GSConv blocks, the resulting model is lightweight yet more accurate. The results demonstrate that the proposed model outperforms several state-of-the-art models. The model was trained and tested on a personal computer using a dataset of 7,000 annotated images from six different types of ships. Despite the model's significant advancements, the authors recommend future research to improve the detection of large overlapping vessels and to embed the application on mobile platforms.

In contrast to the aforementioned studies, this article compares various state-of-the-art OD models, comparing their detection performance, speed, and resource consumption on a Raspberry Pi. The goal is to identify the optimal model for constructing an embedded maritime surveillance solution that balances precision, speed, and resource usage.

Regarding the first related work, which uses different expensive edge devices equipped with powerful computational capabilities to analyze FPS, this article analyses the performance of various models with and without quantization, focusing on achieving the best performance on a cheap and resource-limited device. Moreover, the dataset employed included objects other than ships. The second related work focuses on data clustering techniques to improve YOLOv4's performance. The model is trained and validated on powerful cloud computing resources without dealing with the model's deployment on embedded devices. In contrast, this study targets a more restricted device, achieving scalable and good performance.

While the third related work focuses on model architecture enhancements and accuracy improvements using high-powered personal computers, our research emphasizes the practical application of OD models on resource-constrained devices, balancing speed, precision, and resource consumption. Additionally, this work differs from related ones in that the dataset used in this study is novel and contains only annotated images of ships, which can help further research studies.

III. Methodology

This section presents the methodology employed to select and compare the models. It describes the database used, motivates the choice of models, the device used, and the metrics adopted for comparing the models. All the code related to this study is available on github[1].

A. Dataset

The annotated dataset created especially for this study is stored at the Roboflow Universe [14] and contains 4998 images and ten ship classes. The classes are Bulk Carrier, Container Ship, General Cargo, Oil Product Tanker, Passenger Ship, Tanker, Trawler, Tug, Vehicle Carrier, Yacht, and Background. The number of samples for each class is balanced. The dataset was divided into 70% images for training, 20% for validation, and 10% for testing. The size of the images is 416 pixels in width and height.

B. Models

This section briefly describes the five models assessed in this work. In this study, we carefully selected five object detection models, each representing a balance between complexity, size, and efficiency, to evaluate their performance on a resource-limited device. Our goal was to assess a range of models from the most lightweight and efficient to those that are more complex and state-of-the-art, ensuring a comprehensive analysis for embedded maritime surveillance applications.

[1]https://github.com/HyAgOsK/DetecShipsAtechMultpleModels (available after publication)

1) FOMO: It is a lightweight OD model designed by Edge Impulse for low-capacity devices. Utilizing the MobileNetV2 architecture, it classifies 8x8 pixel cells with 1x1 convolution and softmax activation, identifying objects by their centroids instead of using bounding boxes [15]. This approach simplifies the tracking of similarly sized, well-spaced objects. However, it is not effective for varying object sizes. Additionally, the mAP metric can not be computed due to its lack of bounding box predictions [16].

2) MobileNetV2 SSD FPN Lite 320x320: It is a single-stage OD model designed for mobile and embedded devices [17]. It is a lightweight CNN known for its efficiency in terms of speed and size. This model employs the FPN technique, which improves accuracy by combining features from different network layers. The "Lite" in its name indicates that this is a smaller and faster version of FPN designed for resource-constrained devices [18].

3) YOLOv5n: It was developed by Ultralytics and is a scalable, state-of-the-art OD model known for its exceptional speed and accuracy in object detection [19]. Built on PyTorch and trained on the COCO 2017 dataset, it offers versatile export options for various applications [19]. The YOLOv5n version is a smaller, lighter variant with only 1.9 million parameters, making it suitable for ultra-light mobile solutions and resource-constrained embedded devices.

4) YOLOv8n: It was also developed by Ultralytics and is a state-of-the-art object detection model built on YOLOv5 with improved feature extraction [20]. The lightweight nano version, YOLOv5n, with approximately 3.5 million parameters, is suitable for mobile and embedded devices. YOLOv8 utilizes anchor boxes for accurate bounding box predictions and supports object tracking, making it ideal for surveillance and traffic monitoring while balancing accuracy and efficiency across various tasks.

5) YOLOv10n: It was developed by the Multimedia Intelligence Group at Tsinghua University. It is the latest iteration of the YOLO series, featuring notable advancements in size, accuracy, and speed [21]. It presents a significant leap forward with non-maximum suppression (NMS)-free training, spatial-channel decoupled downsampling, and large-kernel convolutions. Although supporting multiple export formats like ONNX and CoreML, YOLOv10 still lacks support for TensorFlow Lite (TFLite), limiting its applicability in certain mobile and embedded applications.

It is important to highlight that these models were trained on the cloud, either on Edge Impulse [15] or on Google Colaboratory [22], and inference is conducted on a Raspberry Pi device. The FOMO model was trained on Edge Impulse, while MobileNetv2 SSD FPN Lite, YOLOv5n, YOLOv8n, and YOLOv10n models were trained on Google Colaboratory. After training, the saved models are loaded into the Raspberry Pi for validation (i.e., inference).

The selected models were also trained using quantization to reduce their computational burden on resource-constrained devices such as the Raspberry Pi 4. Quantization decreases model size by converting parameters from high-precision types (32-bit floating-point) to lower-precision types (16-bit floating-point or 8-bit integers). This optimization reduces CPU and RAM usage and increases FPS during detections [23].

Figure 1. Flowchart of study's setup and its components.

C. Device

For this study, we used a Raspberry Pi 4 Model B as a proof of concept. The device is configured with 4 GB of LPDDR4 RAM and a quad-core 64-bit ARM Cortex-A72 CPU running at 1.5 GHz. It is powered by a 5V / 3A USB Type-C power supply. An attached TEDGE 720p High Definition (HD) camera, connected via a USB-B cable, captures video in real-time and sends the data to the Raspberry Pi 4B for processing by the OD models. All processing is conducted on the CPU, as the device lacks a dedicated graphics card or hardware accelerator for image processing. Figure 1 illustrates the setup configuration.

D. Metrics

The selected models were evaluated based on several key performance metrics, including inference speed (measured in FPS), CPU and RAM usage, model size, mAP, precision, recall, and F1-score. These criteria provide a comprehensive assessment of each model's efficiency and accuracy, particularly in the context of deployment on resource-constrained devices. These metrics were obtained by running the saved models on the Raspberry Pi 4. FPS is calculated by measuring the total inference time for each image. CPU and RAM usage were monitored in real-time through the Linux terminal using the *htop* command. While mAP, precision, recall and F1-score were measured based on the test set.

IV. Results and Discussions

In this section, we present the results of our comparative analysis, focusing on the impact of model quantization on resource utilization and performance metrics. Table I provides a comprehensive overview of each model's quantization level and its effects on CPU and RAM consumption, FPS, and accuracy measures such as mAP, precision, recall, and F1-score[2]. The data highlights how varying levels of model quantization influence resource efficiency and detection performance, with notable differences in RAM and CPU usage, and FPS. This analysis is crucial for understanding the trade-offs involved in deploying object detection models on resource-constrained devices.

[2]As of the writing of this work, YOLOv10 does not support quantization yet.

Table I
MODELS' METRICS.

Model	Type	FPS	CPU	RAM	mAP	Precision	Recall	F1-score	Size
FOMO	float32	50	13%	150 MB	-	58%	74%	64%	83 kB
	int8	**60**	**12%**	**110 MB**	-	57%	72%	64%	**55 kB**
MobileNetV2 SSD FPN Lite 320x320	float32	5	32%	500 MB	84%	90%	82%	86%	10.9 MB
	int8	7	30%	400 MB	82%	85%	82%	83%	3.58 MB
YOLOv5n	float32	2	71%	860 MB	91%	**93%**	81%	87%	3.63 MB
	float16	5	65%	800 MB	91%	92%	80%	86%	3.48 MB
	int8	6	15%	210 MB	83%	80%	70%	75%	2 MB
YOLOV8n	float32	2.3	25%	450 MB	**93.1**%	89%	88%	**88%**	11.89 MB
	float16	2.9	23%	410 MB	92%	85%	88%	86%	5.99 MB
	int8	3	20%	400 MB	91%	86%	89%	87%	3.1 MB
YOLOV10n	float32	1	90%	500 MB	92.7%	87%	**90%**	**88%**	5.6 MB

Based on the results presented in Table I, several key observations can be made regarding FPS and resource usage. FOMO achieves the highest FPS, especially in its int8 quantized form (60 FPS), while maintaining the lowest CPU (12%) and RAM usage (110 MB). This makes it extremely efficient for resource-constrained devices and applications requiring high FPA, such as real-time ones. In comparison, MobileNetV2 SSD FPN Lite 320x320 has moderate FPS (7 for int8), with relatively higher CPU (30%) and RAM (400 MB) usage compared to FOMO. It performs well in terms of mAP (82%) and precision (85%). YOLOv5n offers lower FPS (6 for int8) but achieves high mAP (91% for float32/float16) and the highest precision (93% for float32). The int8 version significantly reduces CPU (15%) and RAM (210 MB) usage, making it more efficient. YOLOv8n provides a balanced performance with moderate FPS (3 for int8) and lower CPU (20%) and RAM (400 MB) usage than YOLOv5n. It maintains high mAP (93.1% and 92% for float32/float16) and F1-score (88% for float32). YOLOv10n exhibits the lowest FPS (1 for float32) and the highest CPU usage (90%), but achieves high mAP (92.7%) and F1-score (88%). It is less efficient in terms of resource consumption, making it less suitable for highly resource-constrained environments.

It should be underscored that part of the consumption of resources (i.e., CPU and RAM) is due to video processing, especially because an HD camera was employed in this study. Therefore, it might be reduced if cameras with lower resolution are used. However, there might be a trade-off between image resolution and the precision achieved by the OD models.

Regarding accuracy and precision, YOLOv8n and YOLOv10n stand out with the highest mAP (93.1-92.7%) and precision (89-87%), demonstrating superior detection accuracy. YOLOv5n also shows strong performance in accuracy metrics with a high mAP (91% for float32) and F1-score (87% for float32). FOMO and MobileNetV2 SSD FPN Lite 320x320 exhibit lower performance across mAP, precision, recall, and F1-score metrics.

Quantization generally improves FPS and reduces resource consumption (CPU and RAM) at the expense of slight reductions in mAP, precision, recall, and F1-score. This trade-off is particularly evident in models like YOLOv5n and YOLOv8n, where int8 quantization significantly enhances efficiency.

In terms of model size, FOMO is extremely compact, especially in its int8 version (55 kB), making it ideal for devices with very limited storage. The YOLO models, even in their quantized forms, maintain reasonable sizes (2-3.1 MB), effectively balancing performance and storage requirements. There is a clear correlation between model size and FPS: the larger the model, the slower its processing speed. This connection arises from the number of parameters each model contains, which dictates the amount of computation required. Therefore, models with more parameters impose a higher processing burden.

Among all assessed models, YOLOv5n offers a well-rounded combination of moderate to high accuracy, efficient resource usage, and acceptable FPS, making it the most consistent model. YOLOv8n is the second most consistent model and has the additional advantage of also being an object tracker, which renders it an important part of future intelligent maritime surveillance solutions.

However, the choice of model depends on the specific application requirements. FOMO excels in environments where speed and resource efficiency are critical, while YOLO models (especially YOLOv5n and YOLOv10n) offer superior accuracy for applications that can accommodate higher resource consumption. Quantization proves to be a valuable technique for enhancing efficiency across all models where it is possible.

Figures 2, 3, 4, and 5 show the Precision-Recall curves with the mAP metric for an Intersection over Union (IoU) of 0.5 for all models except FOMO in the float32 version. The curve is not available for the FOMO model as it does not provide bounding boxes [24].

As observed in these figures, although some models perform slightly better than others, the worst-performing classes are Oil Product Tanker and Tanker. The similarity between these two classes poses a challenge for the models, making it difficult to distinguish and classify them accurately. To address the misclassification of these two similar classes, some strategies can be employed. Increasing training data and using data augmentation can improve model differentiation. Post-processing techniques, such as model ensembles, can refine predictions.

On the other hand, the best-performing classes are container ship, passenger ship, trawler, tug, vehicle carrier, and yacht. Their curves are well above the average of all individual

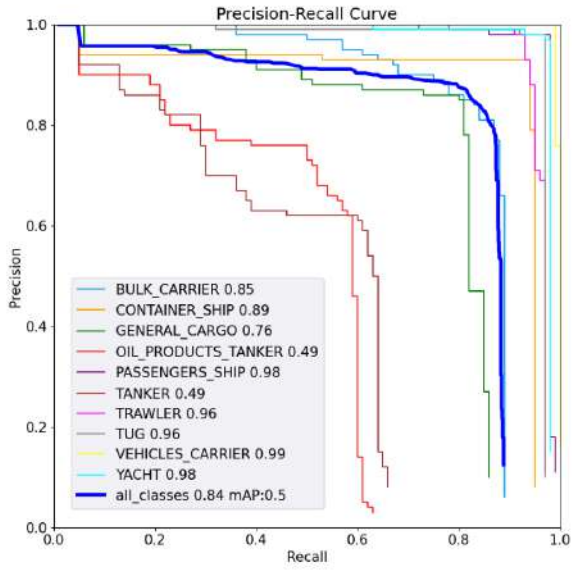

Figure 2. Precision-recall curve of MobileNetV2 SSD FPN Lite 320x320 for its float32 version.

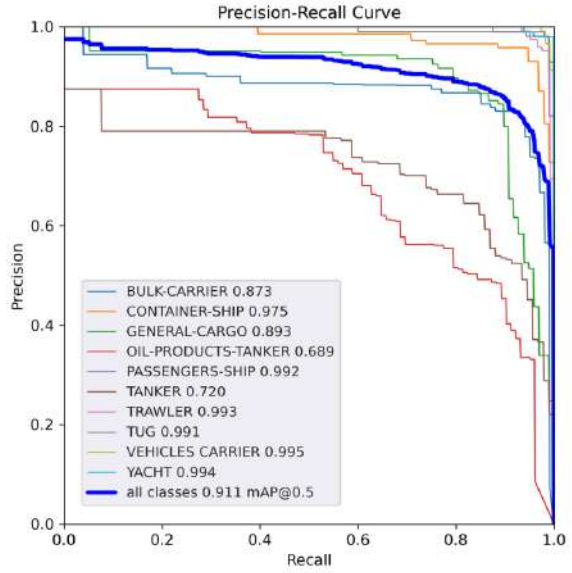

Figure 3. Precision-recall curve of YOLOv5n for its float32 version.

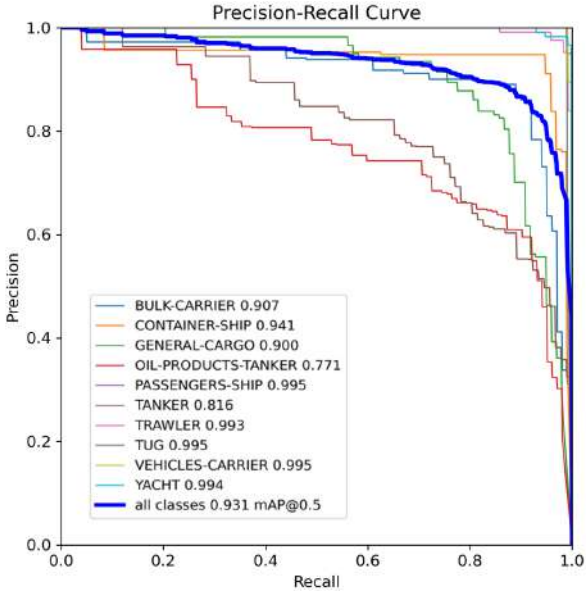

Figure 4. Precision-recall curve of YOLOv8n for its float32 version.

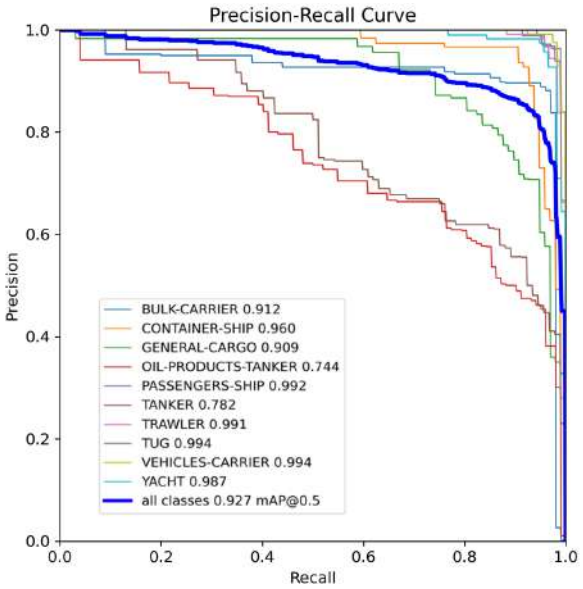

Figure 5. Precision-recall curve of YOLOv10n for its float32 version.

precision-recall curves while the classes bulk carrier and general cargo perform on average. For the latter two classes, the same strategies mentioned before can be tried.

For this study, maritime surveillance applications do not require high FPS rates due to the slow nature of most vessels. YOLOv5n and YOLOv8n are the most suitable models for vessel detection. This is attributed to their high accuracy, reasonable FPS rate, and moderate resource consumption. However, YOLOv8n is favored because it offers an additional advantage: the possibility of tracking objects with state-of-the-art algorithms.

V. Conclusions

This study conducted a comparative analysis of five prominent object detection models (FOMO, MobileNetV2 SSD, YOLOv5n, YOLOv8n, and YOLOv10n) to determine their suitability for IoT-based maritime surveillance applications. By evaluating their performance in terms of accuracy, inference speed, and resource consumption, we identified models that best meet specific operational requirements.

Our findings highlight that FOMO is highly efficient in terms of FPS (up to 60 FPS) and resource consumption, making it ideal for applications requiring real-time processing with very limited computational resources, though it sacrifices

some accuracy and does not support bounding box predictions. MobileNetV2 SSD FPN Lite strikes a balance between speed and accuracy, offering moderate FPS (7 for int8 quantized version) and good mAP (82%), making it suitable for resource-constrained devices where moderate accuracy is acceptable.

YOLOv5n achieves high accuracy (mAP of 91% for float32) with a reasonable trade-off in FPS and resource consumption when quantized. It offers a balanced approach, suitable for scenarios requiring both precision and efficiency. YOLOv8n delivers high accuracy (mAP of 93.1% for float32) with moderate resource consumption, making it ideal for applications where accuracy is paramount and resources allow for slightly higher computational demand.

YOLOv10n shows the highest accuracy and F1-score and slightly lower mAP than YOLOv8n. However, it comes at the expense of significantly higher resource consumption and lower FPS, which might limit its use in highly resource-constrained environments. Overall, YOLOv5n emerged as the most balanced model, providing a good compromise between accuracy, speed, and resource usage. YOLOv8n is also a strong contender, especially for applications where tracking capabilities are beneficial, such as maritime, vehicle, and pedestrian surveillance and smart cities. Quantization proved effective in enhancing the efficiency of all applicable models, making them more suitable for deployment on resource-limited IoT devices.

Future research should focus on developing an embedded intelligent maritime solution and further optimizing these models for embedded systems, exploring additional techniques such as pruning and knowledge distillation to reduce computational demands. Additionally, expanding the dataset with more diverse and complex maritime scenarios will help refine these models further to ensure robust performance in real-world applications. Moreover, libraries such as PyArmNN [25], which offers an API optimized to run neural network models on ARM CPUs, and hardware accelerators, such as Google's Coral USB [26], can help reduce the processing time, increasing the processed FPS.

VI. Acknowledgments

This work was partially funded by CNPq (Grant Nos. 403612/2020-9, 311470/2021-1, and 403827/2021-3), by Minas Gerais Research Foundation (FAPEMIG) (Grant Nos. APQ-00810-21 and APQ-03162-24) and by the projects XGM-AFCCT-2024-2-5-1 and XGM-AFCCT-2024-9-1-1 supported by xGMobile – EMBRAPII-Inatel Competence Center on 5G and 6G Networks, with financial resources from the PPI IoT/Manufatura 4.0 from MCTI grant number 052/2023, signed with EMBRAPII.

References

[1] K. Wang, M. Liang, Y. Li, J. Liu, and R. W. Liu, "Maritime traffic data visualization: A brief review," in *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*, pp. 67–72, IEEE, 2019.

[2] E. Teixeira, B. Araujo, V. Costa, S. Mafra, and F. Figueiredo, "Literature review on ship localization, classification, and detection methods based on optical sensors and neural networks," *Sensors*, vol. 22, no. 18, 2022.

[3] R. d. L. Rocha and F. A. P. de Figueiredo, "Beyond land: A review of benchmarking datasets, algorithms, and metrics for visual-based ship tracking," *Electronics*, vol. 12, no. 13, 2023.

[4] E. H. Teixeira, S. B. Mafra, and F. A. P. d. Figueiredo, "Inatechships: A validation study of a novel ship dataset through deep learning-based classification and detection models for maritime applications," ResearchGate Preprint, 2024.

[5] M. R. Cruz, E. H. Teixeira, S. Mafra, and F. A. P. d. Figueiredo, "A multi-faceted approach to maritime security: Federated learning, computer vision, and iot in edge computing," 2023.

[6] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103812, 2023.

[7] A. M. Rekavandi, L. Xu, F. Boussaid, A.-K. Seghouane, S. Hoefs, and M. Bennamoun, "A guide to image and video based small object detection using deep learning: Case study of maritime surveillance," *arXiv preprint arXiv:2207.12926*, 2022.

[8] J. Becerra, A. Ariza, and L. C. Gamarra-Amaya, "Use of open-source satellite data to combat organized crime case study: Detection of vessels associated with drug-trafficking," in *Space Fostering Latin American Societies: Developing the Latin American Continent Through Space, Part 2*, pp. 67–86, Springer, 2021.

[9] A. Sepehri, H. R. Vandchali, A. W. Siddiqui, and J. Montewka, "The impact of shipping 4.0 on controlling shipping accidents: A systematic literature review," *Ocean engineering*, vol. 243, p. 110162, 2022.

[10] D. Situnayake and J. Plunkett, *AI at the Edge*. " O'Reilly Media, Inc.", 2023.

[11] D. Heller, M. Rizk, R. Douguet, A. Baghdadi, and J.-P. Diguet, "Marine objects detection using deep learning on embedded edge devices," in *2022 IEEE International Workshop on Rapid System Prototyping (RSP)*, pp. 1–7, IEEE, 2022.

[12] B. Wang, B. Han, and L. Yang, "Accurate real-time ship target detection using yolov4," in *2021 6th International Conference on Transportation Information and Safety (ICTIS)*, pp. 222–227, 2021.

[13] X. Zhao and Y. Song, "Improved ship detection with yolov8 enhanced with mobilevit and gsconv," *Electronics*, vol. 12, no. 22, p. 4666, 2023.

[14] H. Vieira, "detectionship dataset." https://universe.roboflow.com/hyago-vieira/detectionship , apr 2024. visited on 2024-04-15.

[15] V. Janapa Reddi, A. Elium, S. Hymel, D. Tischler, D. Situnayake, C. Ward, L. Moreau, J. Plunkett, M. Kelcey, M. Baaijens, *et al.*, "Edge impulse: An mlops platform for tiny machine learning," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[16] L. Boyle, N. Baumann, S. Heo, and M. Magno, "Enhancing lightweight neural networks for small object detection in iot applications," in *2023 IEEE SENSORS*, pp. 01–04, 2023.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[18] M. G. Naftali, J. S. Sulistyawan, and K. Julian, "Comparison of object detection algorithms for street-level objects," *arXiv preprint arXiv:2208.11315*, 2022.

[19] G. Jocher, "YOLOv5 by Ultralytics," May 2020.

[20] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023.

[21] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "Yolov10: Real-time end-to-end object detection," *arXiv preprint arXiv:2405.14458*, 2024.

[22] E. Bisong and E. Bisong, "Google colaboratory," *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pp. 59–64, 2019.

[23] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks," *arXiv preprint arXiv:2205.07877*, 2022.

[24] S. Hymel, C. Banbury, D. Situnayake, A. Elium, C. Ward, M. Kelcey, M. Baaijens, M. Majchrzycki, J. Plunkett, D. Tischler, *et al.*, "Edge impulse: An mlops platform for tiny machine learning," *arXiv preprint arXiv:2212.03332*, 2022.

[25] M. Abellán, S. Cuenca-Asensi, and D. Gutiérrez, "Acceleration of object recognition algorithm on embedded platform,"

[26] A. Ghosh, S. A. Al Mahmud, T. I. R. Uday, and D. M. Farid, "Assistive technology for visually impaired using tensor flow object detection in raspberry pi and coral usb accelerator," in *2020 IEEE Region 10 Symposium (TENSYMP)*, pp. 186–189, IEEE, 2020.