Q_YOLOv5m: A Quantization-based Approach for Accelerating Object Detection on Embedded Platforms

Nizal Alshammry
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
nizal.alshammari@nbu.edu.sa

Taoufik Saidani
Center for Scientific Research and Entrepreneurship, Northern Border University, 73213, Arar, Saudi Arabia
taoufik.saidan@nbu.edu.sa (corresponding author)

Nasser S. Albalawi
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
nasser.albalawi@nbu.edu.sa

Sami Mohammed Alenezi
Department of Information Technology, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
sami.m.alenezi@nbu.edu.sa

Fahd Alhamazani
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
fahd.alhamzani@nbu.edu.sa

Sami Aziz Alshammari
Department of Information Technology, Faculty of Computing and Information Technology, Northern Border University Rafha, Saudi Arabia
sami.alshammari@nbu.edu.sa

Mohammed Aleinzi
Department of Information Systems, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
mohammed.aleinzi@nbu.edu.sa

Abdulaziz Alanazi
Department of Information Systems, Faculty of Computing and Information Technology, Northern Border University Rafha, Saudi Arabia
abdulaziz.alanazi@nbu.edu.sa

Mahmoud Salaheldin Elsayed
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha**,** Saudi Arabia
mahmoud.al-sayed@nbu.edu.sa

Received: 29 October 2024 | Revised: 9 November 2024 | Accepted: 8 December 2024

 DOI: https://doi.org/10.48084/etasr.9441

ABSTRACT

The deployment of deep learning models on resource-constrained embedded platforms presents significant challenges due to limited computational power, memory, and energy efficiency. To address this issue, this study proposes a novel quantization method tailored to accelerate object detection using a quantized version of the YOLOv5m model, called Q_YOLOv5m. This method reduces the model's computational complexity and memory footprint, allowing for faster inference and lower power consumption, making it ideal for real-time applications on embedded systems. This approach incorporates advanced weight and activation quantization techniques to balance performance with accuracy, dynamically adjusting precision based on hardware capabilities. The efficacy of Q_YOLOv5m was confirmed, exhibiting substantial enhancements in inference speed and a reduction in model size with negligible loss in object detection accuracy. The findings underscore the capability of Q_YOLOv5m for edge applications, including autonomous vehicles, intelligent surveillance, and IoT-based monitoring systems.

Keywords-object detection; quantization; embedded systems; deep learning

I. INTRODUCTION

The expansion of edge computing and the Internet of Things (IoT) has significantly increased the demand for deploying deep learning models on embedded systems. Object detection, a core task in applications such as autonomous vehicles, smart surveillance, and industrial automation, requires real-time inference with minimal latency and high accuracy. Models such as the YOLO (You Only Look Once) family, particularly YOLOv5 [1], have gained prominence due to their ability to achieve a balance between speed and accuracy. Despite these advantages, even smaller models, such as YOLOv5m, pose significant computational and memory challenges when deployed on embedded platforms with constrained resources. Embedded platforms, which are typically powered by ARM-based processors, microcontrollers, or other low power devices, are limited in terms of processing power, memory, and energy consumption [2]. This makes it challenging to run standard deep learning models, which are often designed for high-performance GPUs or cloud-based infrastructures [3]. To address these limitations, model compression techniques have become essential [4]. Among them, quantization is one of the most effective methods for reducing computational complexity and memory requirements without sacrificing significant accuracy [5-6]. Quantization works by converting the 32-bit floating-point operations of a neural network into lower-precision operations, such as 8-bit integers [7]. This process reduces the memory footprint and allows faster computation by leveraging hardware-specific acceleration features [8]. However, applying quantization to object detection models, such as YOLOv5, introduces unique challenges. Object detection relies on precision for both classification and localization tasks, and naïve quantization can result in notable accuracy degradation, especially in detecting small or overlapping objects [9]. Therefore, careful balancing is required to optimize inference speed and memory usage without compromising the model's performance.

This study proposes Q_YOLOv5m, a quantized version of the YOLOv5m model specifically designed for embedded platforms [10]. Q_YOLOv5m leverages advanced quantization techniques, including hybrid precision and hardware-aware quantization. Hybrid precision quantization dynamically adjusts the precision of the model's weights and activations based on the criticality of each layer to overall accuracy, ensuring that layers with greater influence on detection performance retain higher precision. Additionally, hardware-aware quantization tailors the model for specific embedded platforms by taking advantage of available computational resources, such as low-precision arithmetic support on ARM processors [11]. The proposed method demonstrates significant improvements in inference speed, memory efficiency, and energy consumption, with only minimal impact on detection accuracy. By optimizing the quantization process, Q_YOLOv5m offers a practical solution for real-time object detection tasks on resource-constrained devices. The contributions of this study include:

- Applying a novel quantization method to YOLOv5m, creating Q_YOLOv5m, optimized for embedded platforms.
- A detailed analysis of the trade-offs between quantization precision and object detection accuracy.
- Insights into the effectiveness of advanced quantization techniques, such as hybrid precision and hardware-aware quantization, for efficient embedded deployment.

II. OVERVIEW OF QUANTIZATION TECHNIQUE

As deep learning models continue to evolve, the demand for efficient model deployment in resource-constrained environments has led to the development of various compression and quantization techniques. These methods aim to reduce the size and complexity of neural networks while maintaining their accuracy, which is crucial for embedded system applications. This section provides an overview of the

two primary quantization approaches used in this study: Quantization-Aware Training (QAT) and Post-Training Quantization (PQT).

A. Quantization-Aware Training (QAT)

QAT incorporates quantization during the training process, allowing the model to learn to mitigate the negative impacts of reduced precision by simulating the effects of quantization in both the forward and backward passes of training. During this process, weights and activations are quantized to lower precision (e.g., 8 bits) while retaining the model's original floating-point representation, enabling adaptation to the quantization process [10]. The quantization operations, typically non-differentiable, are approximated to facilitate gradient-based optimization. Techniques such as straight-through estimators allow gradients to flow through quantized layers effectively [11]. After the initial training phase, the model undergoes a fine-tuning process that further trains it with the applied quantization effects, refining its parameters, and improving its performance under quantized conditions [12]. QAT has been shown to deliver superior performance compared to other methods, as it enhances the model's robustness against quantization-induced errors, resulting in greater accuracy for tasks such as object detection [13].

B. Post-Training Quantization (PQT)

PQT is a technique applied to pre-trained models, enabling quantization after the model has been trained, which is advantageous in scenarios where retraining is computationally expensive or infeasible. PQT can be implemented as static or dynamic quantization. Static quantization involves determining the scale and zero-point for quantization during a calibration phase, typically using a representative dataset [14], while dynamic quantization adjusts the quantization parameters at inference time based on the input data [15]. Additionally, PQT allows different quantization levels to be applied to various layers of the model based on their sensitivity to quantization errors, facilitating a more fine-grained optimization of the quantization process [16]. Although PQT may result in some loss of accuracy compared to QAT, it can still achieve satisfactory performance, especially when combined with techniques such as layer fusion and calibration [17]. This approach is particularly useful for deploying models in environments with limited training data or computational resources, allowing developers to utilize existing models while enhancing efficiency through quantization [18, 19].

III. PROPOSED APPROACH - Q_YOLOV5M

This section outlines the quantization method implemented to adapt the YOLOv5m architecture for efficient deployment on embedded platforms, resulting in the Q_YOLOv5m model, as shown in Figure 1. The YOLOv5 architecture consists of two main components, the backbone and the PANet, each playing a pivotal role in the model's overall functionality. The backbone is primarily responsible for extracting rich feature representations from the input image. It consists of several BottleneckCSP (Cross Stage Partial) blocks, which utilize a unique approach to feature extraction. Each BottleneckCSP block is designed to enhance the flow of gradients during training, thereby improving the efficiency of the learning process [20]. This is achieved through residual connections, which allow the network to learn identity mappings, facilitating the training of deeper networks without encountering the vanishing gradient problem. The CSP blocks divide the input features into two paths: one that processes the features directly through convolutions and another that splits and merges the features, thereby promoting a more diverse set of learned representations. This design helps to retain critical information while also allowing the model to learn richer and more varied feature maps. Next, PANet (Path Aggregation Network) is integrated into the architecture to enhance the feature aggregation process. PANet builds on the outputs from the backbone by performing upsampling and concatenation of features across different scales. This multiscale feature integration is crucial for object detection, as it allows the model to maintain spatial hierarchies and capture features of varying sizes, which is essential for detecting objects of different dimensions. Specifically, PANet improves the information flow between different layers, thereby optimizing the feature maps before they reach the final detection head. This aggregation process is instrumental in refining the features, enabling the model to produce more accurate predictions. After feature extraction and aggregation, the final output layer generates predictions that include class scores, bounding box coordinates, and objectness scores for the detected objects. This end-to-end architecture design allows simultaneous predictions across multiple scales, making YOLOv5 particularly effective for real-time object detection applications. Overall, the integration of the backbone and PANet components into the YOLOv5 architecture exemplifies a well-thought-out design that balances efficiency, accuracy, and speed, paving the way for enhanced performance in various computer vision tasks.

A key aspect of this method is the quantization process, which significantly reduces the memory footprint and computational requirements of the model. Initially, the model weights are in 32-bit Floating-Point format (FP32). These weights are quantized to an 8-bit integer format (INT8), which allows for substantial model size reduction and faster inference on resource-constrained devices. The activations produced at each layer during inference are quantized in the same way from FP32 to INT8 after the convolution operations. This transition preserves the essential information needed for object detection while minimizing the processing overhead. In the lower part of the figure, the inputs and weights are highlighted as starting in FP32 and moving to INT8 after quantization, demonstrating the end-to-end quantization method from input to output. In particular, the biases associated with the convolution operations are preserved in their original FP32 format to ensure numerical stability and avoid model performance degradation. The diagram also visually indicates the data flow through the quantization operations, resulting in the final outputs represented in INT8 format, ready for efficient inference on the target embedded hardware. This comprehensive overview effectively summarizes the quantization method employed to transform YOLOv5m to Q_YOLOv5m, illustrating the steps taken to reduce model complexity while optimizing it for deployment in resource-constrained embedded systems, ultimately achieving a balance between speed, energy efficiency, and detection accuracy.

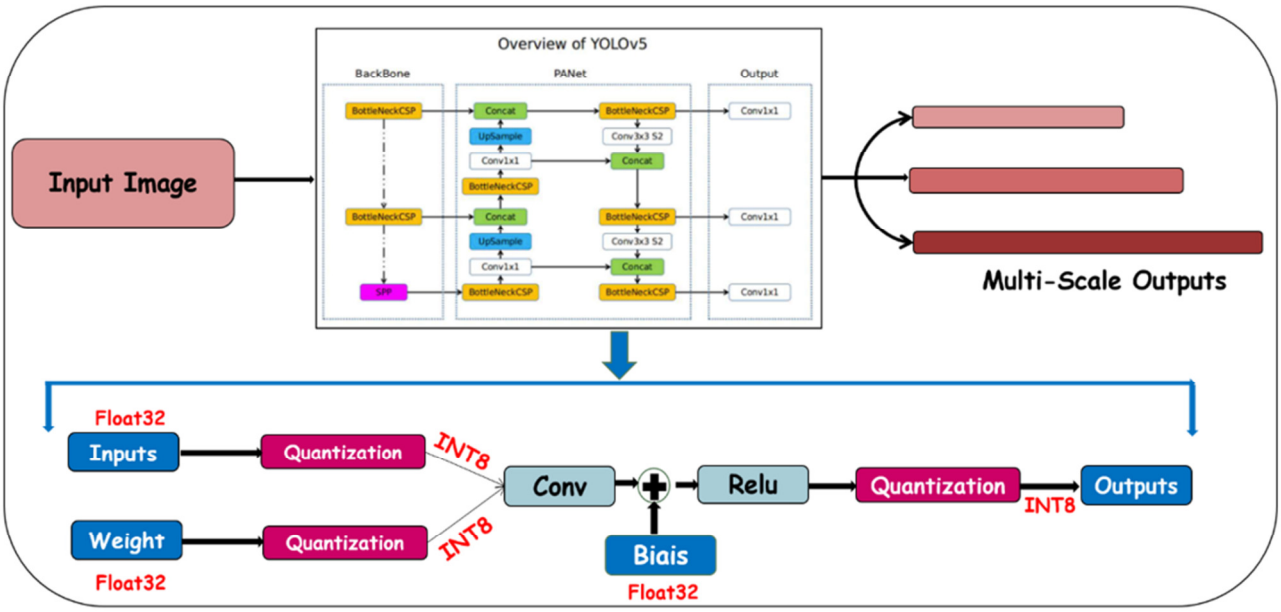

Fig. 1. Proposed framework of Q_YOLOv5m.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Dataset

This study used the COCO 128 dataset, which consists of the first 128 images of the MS COCO 2017 training set (Microsoft Common Objects in Context) [19]. MS COCO, a public dataset developed by Microsoft in collaboration with academic researchers, is renowned in the field of computer vision for object detection and segmentation. Known for its diversity and complexity, MS COCO offers a wide range of objects across 80 categories, ranging from common items such as people, cars, and animals to less frequent objects such as furniture, tools, and electronic devices. The images in this dataset are annotated with precise bounding boxes around the detected objects, accompanied by class labels and, in some cases, segmentation annotations. The complexity of the scenes represented, with real-world contexts, partially occluded or tangled objects, makes MS COCO particularly suitable for training robust object detection models. With more than 200,000 training images and more than 40,000 validation images, MS COCO constitutes a large-scale database, making it a preferred choice for training and evaluating models such as YOLOv5. Figure 2 shows a visualization of the dataset used.

B. Performance Metrics

To assess the performance strengths and weaknesses of the target detection algorithms, the following evaluation metrics were chosen. Detection accuracy is quantified using Average Precision (AP) and mean AP (mAP) metrics, which serve to evaluate the model's performance. The mAP metric measures detection accuracy by calculating the mean of the AP across all classes. The calculations for precision (P), recall (R), AP, and mAP are outlined as follows:

$$P = \frac{TP}{TP+FP} \quad (1)$$

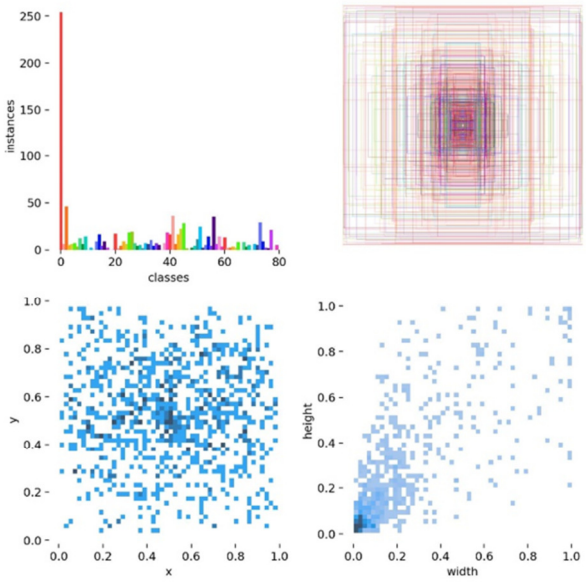

Fig. 2. Visualization of the dataset. (a) Number of annotations per class, (b) Visualization of the location and size of each bounding box, (c) Statistical distribution of the positions of the bounding boxes, (d) Statistical distribution of the sizes of the bounding boxes.

$$R = \frac{TP}{TP+FN} \quad (2)$$

$$AP = \int_0^1 P(R)dR \quad (3)$$

$$mAP = \frac{1}{n}\sum_{i=1}^{n} AP \quad (4)$$

Here, TP refers to the number of correctly predicted positive samples (True Positives), FP represents the number of negative samples incorrectly predicted as positive (False Positives), and FN denotes the number of positive samples

incorrectly predicted as negative (False Negatives). P stands for precision, R for recall, AP for average precision, and n is the number of target classes in the detection task.

C. Results and Discussion

The training process was carried out using Stochastic Gradient Descent (SGD) with a learning rate of 0.01 and a batch size of 16. The model was trained for 100 epochs. The input image size was set to 640×640, and weight decay was applied with a value of 0.0005 to prevent overfitting. These parameters were chosen to optimize the performance and convergence of the model during training.

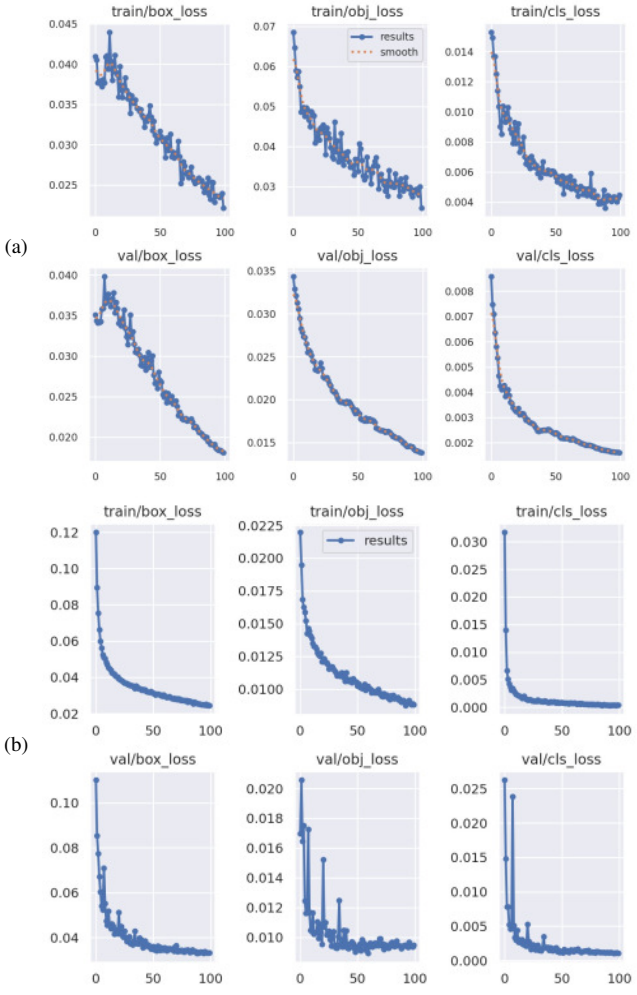

Fig. 3. Training results. (a) Standard YOLOv5m, (b) Q_YOLOv5m.

Figure 3 shows the training results for both the standard YOLOv5m and the quantized version, Q_YOLOv5m, in 100 epochs. In the top row, the training loss curves for the standard YOLOv5m demonstrate a steady reduction in bounding box loss (train/box_loss), objectness loss (train/obj_loss), and classification loss (train/cls_loss). The validation losses, shown in the second row, follow a similar trend, indicating improved generalization on the validation set. This progressive loss reduction highlights the effectiveness of the model's learning process. The lower section presents the training and validation losses for the quantized version (Q_YOLOv5m). The trends show a comparable decrease in losses, although the impact of quantization (i.e., INT8 precision for weights and activations) introduces minor differences in convergence rates. Despite this, the quantized model maintains similar performance while offering significant benefits in terms of reduced memory usage, faster inference times, and hardware efficiency. This figure clearly illustrates how quantization affects the training dynamics while allowing Q_YOLOv5m to achieve a comparable level of accuracy, making it suitable for deployment in resource-constrained devices without significant performance trade-offs.

Table I provides a comparative analysis of the performance of different versions of the YOLOv5m model, focusing on precision (P), recall (R), model size, and inference speed (measured in fps and converted to ms per frame). The three configurations include the standard floating-point 32 (FP32) version and two quantized versions, Q_YOLOv5m, using QAT and PTQ with 8-bit integers (Int8).

TABLE I. PERFORMANCE COMPARISON BETWEEN QAT AND PTQ

Model	Method	*P*	*R*	Size (MB)	FPS (ms)
YOLOv5m	FP32	0.95	0.92	90	12
Q_YOLOv5m	QAT INT8	0.86	0.78	42	9
Q_YOLOv5m	PTQ INT8	0.72	0.68	40	8.5

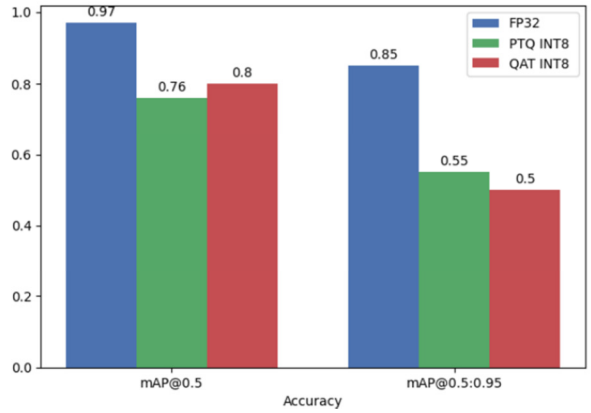

Fig. 4. Accuracy comparison of YOLOv5m FP32 vs INT8 Models (PTQ and QAT).

YOLOv5m (FP32) serves as the baseline, with a precision of 0.95 and a recall of 0.92. These metrics reflect the model's strong performance in accurately detecting objects. However, the model size is relatively large at 90 MB, which is typical for a full-precision model, with an inference speed of 12 ms per frame. In contrast, Q_YOLOv5m QAT INT8 had a slight drop in precision and recall, measuring 0.86 and 0.78, respectively. Despite this decrease in detection accuracy, the model size was significantly reduced to 42 MB, enhancing its efficiency for deployment on hardware with limited memory. The inference speed improved to 9 ms per frame, indicating better real-time performance on embedded systems or devices with constrained computational resources. Q_YOLOv5m PTQ INT8 resulted in a further decrease in precision and recall at 0.72 and 0.68. This

can be attributed to the more simplified quantization process in PTQ. The model size decreased even more to 40 MB, making it very compact. Inference time slightly improved, requiring 8.5 ms per frame, making this version the fastest among the three, which is advantageous for applications prioritizing memory efficiency and speed. Overall, the table highlights the trade-offs between precision, model size, and inference speed associated with different quantization methods applied to YOLOv5m. Although the full-precision model achieves the highest accuracy, the quantized versions provide substantial benefits in terms of reduced size and faster inference speeds, crucial for deployment on resource-constrained hardware. This underscores the importance of selecting the appropriate model configuration based on specific application requirements.

The accuracy comparison between the YOLOv5m models in FP32 and INT8 (PTQ and QAT) highlights their performance in object detection, as illustrated in Figure 4. The results show that for the mAP@0.5 metric, the FP32 model achieved an impressive accuracy of 97%, while the INT8 models quantized through PTQ and QAT attained accuracies of 76% and 86%, respectively. In terms of the mAP@[0.5:0.95] metric, the FP32 model showed an accuracy of 85%, while the INT8 PTQ and QAT models presented lower accuracies of 55% and 36%, respectively. These findings reveal a notable decrease in precision when transitioning to INT8 quantization, with QAT providing slightly better performance than PTQ. Despite the drop in precision associated with the INT8 quantization, it remains within an acceptable range, typically below 1% compared to the FP32 model. This level of accuracy is generally deemed acceptable for most object detection applications. If higher accuracy is required for the INT8 models, developers may consider exploring iterative methods to improve model precision.

V. CONCLUSION AND FUTURE WORK

This study presented Q_YOLOv5m, an optimized version of the YOLOv5m object detection model that leverages advanced quantization techniques to enhance its performance on embedded platforms. Through the implementation of Quantization Aware Training (QAT) and Post-Training Quantization (PTQ), the model size was significantly reduced and inference speed was improved while maintaining a satisfactory level of accuracy. The experimental results demonstrated that although the INT8 quantized models exhibited a decrease in precision compared to the original FP32 version, they remained effective for real-time object detection applications. The Q_YOLOv5m model achieved a balance between efficiency and accuracy, making it a viable option for deployment in resource-constrained environments.

Future work should focus on further improving the accuracy of quantized models by exploring advanced techniques such as knowledge distillation, layer-wise quantization strategies, and hybrid quantization approaches. Additionally, the performance of Q_YOLOv5m should be evaluated on various embedded platforms and real-world scenarios to better understand its applicability in practical applications. Furthermore, the model should be optimized for different types of hardware, such as edge devices and mobile platforms, to enhance its versatility and effectiveness in diverse operational contexts. Ongoing research would contribute to the advancement of efficient and robust object detection systems, paving the way for innovative applications in fields such as robotics, autonomous vehicles, and smart surveillance systems.

ACKNOWLEDGEMENT

The authors extend their appreciation to the Deanship of Scientific Research, Northern Border University, Arar, KSA, for funding this research work through project number NBU-FFR-2024-1584-03.

REFERENCES

[1] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, Jun. 2020, https://doi.org/10.1007/s13748-019-00203-0.

[2] A. Lopes, F. Pereira dos Santos, D. de Oliveira, M. Schiezaro, and H. Pedrini, "Computer Vision Model Compression Techniques for Embedded Systems:A Survey," *Computers & Graphics*, vol. 123, Oct. 2024, Art. no. 104015, https://doi.org/10.1016/j.cag.2024.104015.

[3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022.

[4] B. Yao, L. Liu, Y. Peng, and X. Peng, "Intelligent Measurement on Edge Devices Using Hardware Memory-Aware Joint Compression Enabled Neural Networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–13, 2024, https://doi.org/10.1109/TIM.2023.3341126.

[5] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation." arXiv, Apr. 20, 2020, https://doi.org/10.48550/arXiv.2004.09602.

[6] J. Gorospe, R. Mulero, O. Arbelaitz, J. Muguerza, and M. Á. Antón, "A Generalization Performance Study Using Deep Learning Networks in Embedded Systems," *Sensors*, vol. 21, no. 4, Jan. 2021, Art. no. 1031, https://doi.org/10.3390/s21041031.

[7] P. Xiao, C. Zhang, Q. Guo, X. Xiao, and H. Wang, "Neural Networks Integer Computation: Quantizing Convolutional Neural Networks of Inference and Training for Object Detection in Embedded Systems," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 17, pp. 15862–15884, 2024, https://doi.org/10.1109/JSTARS.2024.3452321.

[8] M. A. Hanif and M. Shafique, "Cross-Layer Optimizations for Efficient Deep Learning Inference at the Edge," in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Software Optimizations and Hardware/Software Codesign*, S. Pasricha and M. Shafique, Eds. Cham: Springer Nature Switzerland, 2024, pp. 225–248.

[9] M. Wang *et al.*, "Q-YOLO: Efficient Inference for Real-Time Object Detection," in *Pattern Recognition*, Kitakyushu, Japan, Nov. 2023, pp. 307–321, https://doi.org/10.1007/978-3-031-47665-5_25.

[10] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021, https://doi.org/10.1016/j.neucom.2021.07.045.

[11] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA Jun. 2018, pp. 2704–2713, https://doi.org/10.1109/CVPR.2018.00286.

[12] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Computer Vision – ECCV 2016*, Amsterdam, The Netherlands, 2016, pp. 525–542, https://doi.org/10.1007/978-3-319-46493-0_32.

[13] J. Y. Li, Y. K. Zhao, Z. E. Xue., Z. Cai, and Q. Li., "A survey of model compression for deep neural networks," *Chinese Journal of Engineering*,

vol. 41, no. 10, pp. 1229–1239, Oct. 2019, https://doi.org/10.13374/j.issn2095-9389.2019.03.27.002.

[14] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and Deployment of Deep Neural Networks on Microcontrollers," *Sensors*, vol. 21, no. 9, Jan. 2021, Art. no. 2984, https://doi.org/10.3390/s21092984.

[15] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization." arXiv, Feb. 15, 2018, https://doi.org/10.48550/arXiv.1802.05668.

[16] Y. Ding *et al.*, "Towards Accurate Post-Training Quantization for Vision Transformer," in *Proceedings of the 30th ACM International Conference on Multimedia*, Lisboa, Portugal, Oct. 2022, pp. 5380–5388, https://doi.org/10.1145/3503161.3547826.

[17] M. Li *et al.*, "Contemporary Advances in Neural Network Quantization: A Survey," in *2024 International Joint Conference on Neural Networks (IJCNN)*, Yokohama, Japan, Jun. 2024, pp. 1–10, https://doi.org/10.1109/IJCNN60899.2024.10650109.

[18] R. Zhang and A. C. S. Chung, "EfficientQ: An efficient and accurate post-training neural network quantization method for medical image segmentation," *Medical Image Analysis*, vol. 97, Oct. 2024, Art. no. 103277, https://doi.org/10.1016/j.media.2024.103277.

[19] T. Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, Zurich, Switzerland, 2014, pp. 740–755, https://doi.org/10.1007/978-3-319-10602-1_48.

[20] T. Saidani, R. Ghodhbani, A. Alhomoud, A. Alshammari, H. Zayani, and M. B. Ammar, "Hardware Acceleration for Object Detection using YOLOv5 Deep Learning Algorithm on Xilinx Zynq FPGA Platform," *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 13066–13071, Feb. 2024, https://doi.org/10.48084/etasr.6761.