Quantized Object Detection for Real-Time Inference on Embedded GPU Architectures

Fatima Zahra Guerrouj[1], Sergio Rodríguez Flórez[2], Abdelhafid El Ouardi[3], Mohamed Abouzahir[4], Mustapha Ramzi[5]
*Université Paris-Saclay, ENS Paris-Saclay, CNRS, SATIE, 91190, Gif-sur-Yvette, France[1,2,3]
Systems Analysis-Information Processing and Industrial Management Laboratory-
Higher School of Technology of Sale, Mohamed V University, Rabat, Morocco[1,4,5]

***Abstract*—Deploying deep learning-based object detection models like YOLOv4 on resource-constrained embedded architectures presents several challenges, particularly regarding computing performance, memory usage, and energy consumption. This study examines the quantization of the YOLOv4 model to facilitate real-time inference on lightweight edge devices, focusing on NVIDIA's Jetson Nano and AGX. We utilize post-training quantization techniques to reduce both model size and computational complexity, all while striving to maintain acceptable detection accuracy. Experimental results indicate that an 8-bit quantized YOLOv4 model can achieve near real-time performance with minimal accuracy loss. This makes it well-suited for embedded applications such as autonomous navigation. Additionally, this research highlights the trade-offs between model compression and detection performance, proposing an optimization method tailored to the hardware constraints of embedded architectures.**

***Keywords*—Object detection model; quantization; embedded architectures; real-time**

I. Introduction

The rapid advancement of artificial intelligence technologies has led to significant growth in object detection models based on deep learning, especially CNN (Convolutional Neural Network), transforming various industries, including autonomous driving, video surveillance, mobile robotics, and intelligent embedded systems [1]. These algorithms facilitate a precise, real-time understanding of environments by detecting and locating objects of interest within video streams. Among these models, the YOLO (You Only Look Once) family, particularly the YOLOv4 version, has emerged as a benchmark due to its remarkable ability to balance high precision, surpassing many of its predecessors on standard evaluation metrics.

While this level of performance is impressive, it also comes with a significant drawback: high computational complexity. Implementing such models requires substantial computing resources, typically provided by high-end GPUs. This poses a considerable challenge when deploying these models on resource-constrained embedded architectures based on CPU-GPUs, like the NVIDIA Jetson Nano that is fitted with a 128-core NVIDIA Maxwell GPU, a 4-core ARM Cortex-A57 MPCore operating at 1.43 GHz, and 4 GB of memory, and are praised for their affordability, compactness, and low power consumption. These devices often lack the hardware capabilities to support demanding models such as YOLOv4 while maintaining mission-sensitive embedded systems' critical latency and power consumption requirements.

In autonomous navigation, the rapid and reliable detection of objects, including vehicles, pedestrians, and cyclists, is essential. The system must make real-time decisions to avoid obstacles or adjust its trajectory, necessitating quick processing, often within 33 ms per image (or more than 30 FPS). To meet these rigorous requirements, various optimization techniques are being explored to reduce the size and complexity of models without significantly sacrificing performance [2].

One key strategy is pruning [3], which systematically removes neural connections or weights that contribute minimally to model outcomes. For example, weights close to zero in dense or convolutional layers are eliminated post-training. This reduction in weight connectivity enhances computational efficiency and substantially decreases the number of calculations required during inference.

Another technique is knowledge distillation [4], which involves transferring knowledge from a high-capacity teacher model to a smaller, more efficient model, the student. This process aims to maintain accuracy while minimizing inference time and computational costs, which is vital for deploying object detection models on embedded systems with limited resources.

Additionally, network architectures can be redesigned to include lighter modules or compact models like MobileNet or Tiny-YOLO [5]. Importantly, quantization involves converting the weights and activations of a floating-point precision array (32 bits) into more compact formats, such as INT8. This conversion reduces the model's memory footprint, accelerates matrix operations, and conserves power consumption. This technique is particularly effective on compatible hardware architectures, enabling swift execution even on architectures like the Jetson Nano. However, this compression may slightly reduce accuracy, necessitating fine-tuning and calibration steps such as quantization-aware training or post-training quantization to achieve an acceptable balance between computational efficiency and detection accuracy [6].

This study investigates, assesses, and enhances the integration of quantized YOLOv4 on a resource-constrained embedded architecture, focusing on critical object detection for autonomous navigation. The objective is to advance the design of intelligent embedded systems that can make reliable, swift, and safe decisions, even within constrained environments.

The paper is organized as follows: Section II reviews the literature on object detection in resource-constrained environments, focusing on quantization and edge deployment.

Section III outlines the evaluation methodology, including the dataset, metrics, and experimental setup. Section IV details the YOLOv4 model optimization process, including conversion and quantization for efficient inference on the Jetson Nano. Section V analyzes results, comparing precision levels and hardware architectures, while Section VI concludes with key findings and future work directions.

II. Related Work

Real-time object detection on ressouce-constrained architectures has emerged as a significant area of research within the broader realm of embedded artificial intelligence. This growing interest stems from the demand for deploying efficient, accurate, scalable vision systems in resource-constrained environments. Recent advancements in deep learning, quantization, and hardware-specific optimizations have facilitated the practical implementation of object detection models on architectures such as NVIDIA Jetson devices, Raspberry Pi boards, and FPGA-based systems. Numerous studies have tackled these challenges using various strategies, including model compression, unsupervised learning, and comparative analyses across different hardware architectures.

The work presented by [7] offers a practical and timely contribution to advanced AI by demonstrating how quantized deep learning models can facilitate efficient, real-time object detection on resource-constrained architectures when paired with hardware acceleration. The study compares a quantized YOLOv3-tiny model on an FPGA-based Zedboard and an FP16-optimized YOLOv7-tiny model on the GPU-powered Jetson Nano. It underscores the trade-offs between power consumption, inference speed, and detection accuracy. Although the Zedboard exhibits very low power consumption, its high inference latency renders it unsuitable for real-time applications, emphasizing further FPGA optimizations. In contrast, the Jetson Nano strikes a commendable balance with 38 FPS and a mean Average Precision (mAP) of 46.3% at just 5.1 W, validating the effectiveness of quantization and GPU acceleration for edge deployment.

Similarly, the study achieved by [8] presents a practical approach to implementing real-time object detection in edge video surveillance systems. The authors tackle the challenges associated with the limited computing power and energy efficiency of edge devices, which are crucial for enabling real-time processing capabilities. The study achieves a notable enhancement in object detection performance for resource-constrained edge devices by utilizing quantized transfer learning with MobileNet V2 SSDs and applying 8-bit quantization. Test results indicate that the Raspberry Pi 5 and the Nvidia Jetson Orin Nano exceed the performance of other devices, with total latencies of 5 ms and 85 ms, respectively, highlighting their effectiveness for real-time applications. The quantized int8 model reaches an accuracy of 80.65% while significantly reducing both memory consumption and latency compared to the unoptimized int32 model.

Further, the research completed by [9] centers on implementing efficient object detection and recognition techniques tailored for resource-constrained embedded systems, utilizing open-source tools such as OpenCV. The authors investigate lightweight deep learning models, including MobileNet-SSD, to achieve real-time performance on devices with limited computational capabilities. By leveraging pre-trained models and optimizing them through quantization techniques, the study illustrates the viability of deploying object detection applications in environments characterized by restricted processing power and resources. Another significant contribution comes from [10], which presents a thorough investigation into deploying a People Search System (PSS) on the Nvidia Jetson Orin AGX architecture, emphasizing model compression techniques to enhance performance on resource-constrained embedded architectures. They implement quantization and pruning techniques alongside L1 regularization to decrease the model size and computational requirements, enabling the real-time processing capabilities crucial for monitoring applications. By executing and assessing the PSS on both GPU and Jetson Orin AGX architectures, the study offers valuable insights into the trade-offs between model accuracy, inference speed, and resource utilization. Additionally, the use of open-source libraries and frameworks highlights the practical applicability of the proposed system.

Additionally, the study by [11] presents a well-executed study focused on optimizing and deploying the YOLOv7 deep learning model for object detection on the NVIDIA Jetson Nano, a cutting-edge low-power AI architecture. The authors successfully refined the YOLOv7 model using TensorRT and quantization techniques to enhance inference speed without compromising detection accuracy. The model achieves an impressive average accuracy of 92.35% and an average processing time of 117.8 ms, underscoring the feasibility of implementing advanced object detection systems on resource-constrained devices. The paper examines the potential and limitations of executing real-time AI workloads on edge architectures by assessing key performance metrics such as speed, accuracy, and resource utilization across various experimental classes and conditions.

Further, the work of [12] presents a robust and well-designed framework for unsupervised object detection in video, specifically targeting real-time performance on low-power embedded systems. It effectively addresses the key challenges of traditional pipelines, including the reliance on extensive labeled datasets and significant computational demands, by utilizing optical flow for motion-based detection and implementing unsupervised clustering to eliminate the necessity for manual annotation. By harnessing the computational power of the NVIDIA Jetson AGX Xavier, the authors adopt a hardware-aware optimization strategy that leverages its heterogeneous processing units (CPU, GPU, and DLA) and incorporates mixed-precision computing (FP32, FP16, INT8). Consequently, the proposed system achieves a remarkable 32.3× speed increase and 23.6× improvement in energy efficiency compared to an unoptimized reference, all while maintaining a competitive mean Average Precision (mAP) of 59.44. These results underscore the framework's suitability for edge computing applications that require stringent performance and energy efficiency.

Lastly, the work of [13] offers a comprehensive, practical comparative analysis of several state-of-the-art neural network models for real-time object detection on low-power edge devices. It provides valuable insights into the trade-offs between accuracy, inference speed, and computational

efficiency. By evaluating models such as MobileNetV2 SSD [14], CenterNet [15], EfficientDet, and various iterations of YOLO (including YOLOv7 Tiny and YOLOv8) [16] on devices like the Raspberry Pi and NVIDIA Jetson Nano, the authors deliver a well-rounded comparison that is relevant to real-world deployment scenarios. Incorporating post-training quantization (PTQ) and quantization-aware training (QAT), along with fine-tuning on a customized dataset, underscores the study's applicability and technical rigor. The recommendations based on specific frames per second (FPS) requirements are beneficial for guiding practitioners in choosing the most suitable model-device combinations to address the various constraints of their applications.

The literature reviewed emphasizes notable advancements in deploying object detection models on edge devices through various optimization techniques, including quantization, pruning, model compression, and architecture refinement. Prior studies have demonstrated that integrating lightweight models with post-training quantization and specific hardware acceleration, especially using architectures like Jetson Nano, Jetson Orin, and Raspberry Pi, can result in efficient real-time inference while maintaining acceptable accuracy trade-offs. Nonetheless, challenges persist in achieving an optimal balance between detection accuracy, inference speed, and memory efficiency, particularly under tight resource constraints.

Building on this foundation, our work focuses on deploying a complete YOLOv4 model recognized for its superior detection quality on the Jetson AGX and Nano utilizing TensorRT-based FP16 and INT8 quantization. This approach expands existing research by illustrating the performance of a more sophisticated model under practical edge conditions, supported by quantitative benchmarks and qualitative validation. The primary contributions of this work are outlined as follows:

1) YOLOv4 on embedded architecture: We demonstrate executing the YOLOv4 object detection model on the resource-constrained Jetson AGX and Nano, addressing real-time performance challenges in embedded environments.

2) Post-training quantization with TensorRT: We compare YOLOv4 quantized in FP32, FP16, and INT8 formats using TensorRT, showing significant improvements in model size and inference speed with minimal accuracy loss

3) Real-world validation: Qualitative analysis in urban settings confirms that the INT8-quantized YOLOv4 model reliably detects cars, cyclists, and pedestrians on a low cost CPU-GPU architecture (Jetson Nano) and Jetson AGX.

4) Guidance for deploying embedded AI: This study offers a reference for deploying deep learning models on embedded systems, highlighting an efficient optimization pipeline and trade-offs in accuracy, speed, and memory usage.

III. Evaluation Methodology

The evaluation of object detection algorithms is crucial for developing and validating computer vision systems, especially in high-stakes applications like autonomous driving. In this section, we outline the methodology used to assess the performance of the YOLOv4 object detection model, detailing the experimental workflow, dataset, evaluation metrics, and hardware architectures employed. We describe the YOLOv4 inference pipeline tailored for deployment in both edge and server environments. To ensure a thorough evaluation under realistic conditions, we utilize the KITTI dataset [18], which is widely recognized as a benchmark for autonomous driving and object detection tasks. Model performance is evaluated using standard metrics, including mean Average Precision (mAP), average Precision (AP) per class, and frames per second (FPS) [19], allowing us to assess both accuracy and real-time processing capabilities. Finally, we present the experimental infrastructure, which consists of a high-performance server for baseline comparisons, and the NVIDIA Jetson Nano, a resource-constrained edge device, to evaluate the feasibility and effectiveness of the model in embedded environments.

A. Detection Model

YOLOv4 (You Only Look Once version 4) is a one-stage object detection model designed to strike an optimal balance between accuracy and real-time performance [20]. Building upon the strengths of its predecessors, YOLOv4 integrates a variety of architectural innovations and training techniques that significantly enhance detection speed and accuracy, making it particularly well-suited for edge deployment and real-time applications.

The software architecture of YOLOv4 consists of several key functional components that contribute to its impressive performance. At the core of the network lies the Backbone, CSPDarknet53, which is tasked with extracting features from input images. The Neck module SPP and PANet enable multi-scale feature fusion and improve the receptive field [21]. Finally, the head includes detection layers for predicting bounding boxes and class probabilities.

The choice of YOLOv4 for this study is based on its demonstrated efficacy in prior research. As mentioned in the work of [22], YOLOv4 outperforms many contemporary models in terms of both accuracy and processing speed, making it an ideal candidate for deployment on high-end, resource-constrained architectures. Its robustness, modularity, and compatibility with optimization techniques such as quantization and pruning further enhance its suitability for edge computing applications. Fig. 1 presents the complete YOLOv4 architecture, detailing the key components, including CBM, CBL, SPP, PANet, and detector heads, providing a comprehensive overview of the model's internal structure.

B. Dataset

Selecting an appropriate dataset is crucial for developing and evaluating object detection algorithms for autonomous vehicles, as it ensures robustness and real-world applicability. Datasets must accurately reflect driving environment complexities, including weather, lighting, and diverse object classes like vehicles and pedestrians. High-resolution images and precise annotations, including bounding boxes and object class labels, are essential.

In this study, we focus on the KITTI dataset, a key resource for autonomous driving evaluation, which includes a variety of real-world scenarios near Karlsruhe, Germany [18]. It features 7,481 images with detailed ground truth labels across different environments, such as freeways and urban streets, and we divided it into a training set (70%, 5,237 images) and a test

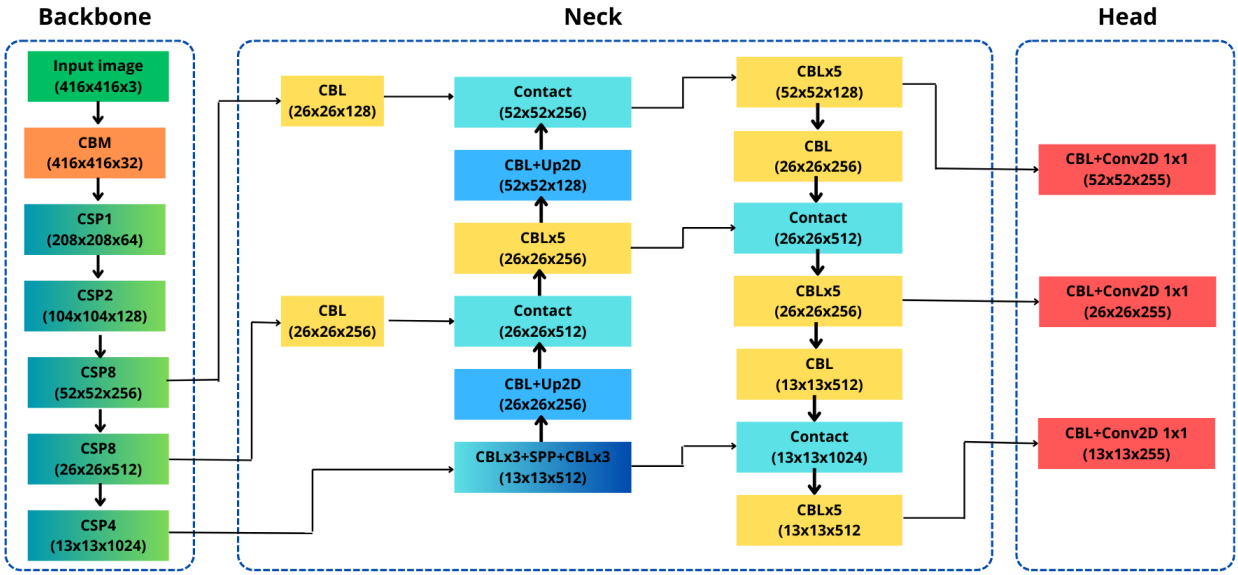

Fig. 1. YOLOv4 architecture [17].

set (30%, 2,244 images). We focus on three key object classes: cars, pedestrians, and bicycles. KITTI also offers established evaluation metrics, allowing for the objective comparison of model performance. Its diverse sensor data and high-quality annotations make it fundamental for advancing object detection algorithms in autonomous driving.

C. Specification of Hardware Architectures

This study utilized a high-performance workstation alongside the embedded architectures to assess and optimize our object detection models. The workstation is equipped with an Nvidia Quadro RTX 6000 GPU that features 24 GB of memory and 4608 CUDA cores, paired with an Intel® Xeon® W-2265 CPU operating at a frequency of 3.50 GHz. It runs on Pop!_OS 20.04 LTS and utilizes version 11.7 of the CUDA compiler tools, creating a robust environment for training and testing deep learning models. Additionally, we employed the Jetson AGX Xavier, which is equipped with a 512-core Volta GPU, an 8-core ARM Carmel CPU running at 2.26 GHz, and 16 GB of memory. On the other hand, we employed the Nvidia Jetson Nano, which is fitted with a 128-core NVIDIA Maxwell GPU, a 4-core ARM Cortex-A57 MPCore operating at 1.43 GHz, and 4 GB of memory. The architectures allow for deploying complex object detection models in resource-constrained environments, particularly suited for real-time applications such as autonomous driving.

D. Evaluation Metrics

Assessing the performance of object detection models necessitates using standardized metrics that quantify both the precision and reliability with which the system identifies and locates objects within an image. Standard evaluation metrics for object detection encompass precision, recall, average precision (AP), mean average precision (mAP), and processing speed [19].

1) Mean average precision: In object detection-based Convolutional Neural Networks (CNNs), mean Average Precision (mAP) is a crucial evaluation metric for assessing the performance of models like YOLOv4. mAP quantifies how effectively the model is able to locate and classify objects within an image accurately. This is especially vital in autonomous vehicle applications, where safety and reliability are paramount. A higher mAP indicates that the model can reliably detect essential objects, such as pedestrians, vehicles, and bicycles, in various scenarios and lighting conditions.

To evaluate the model's precision performance, metrics such as Average Precision, mean Average Precision, Precision, and Recall are employed [23]. These metrics are calculated as follows:

Precision (P): Measures the quality of the model in terms of its ability to detect true positives among all positive predictions [24]. It is defined as follows:

$$P = TP + /(TP + FP) \tag{1}$$

The value ranges from 0 to 1. TP stands for True Positive, and FP for False Positive. Recall (R): Is a quantitative measure of the model's ability to find true positives among all predictions [24]. It is defined as follows:

$$R = TP + /(TP + FN) \tag{2}$$

As with precision, the recall value is also between 0 and 1. FN stands for False Negative.

Average Precision (AP): This measure is commonly used to evaluate the balance between precision and recall in object detection tasks. It measures the model's ability to detect and locate objects in each class accurately [25]. The simplified formula for AP is as follows:

$$AP = \sum(Recall(i) - Recall(i-1)) * Precision(i) \quad (3)$$

Where i iterates over all points where Recall changes, Recall (i) represents Recall at point i, and Precision (i) represents Precision at point i.

A higher AP value indicates better overall detection performance for an object class. An AP of 1 means perfect detection, i.e. the model identifies all objects in this class without any false positives.

Mean Average Precision (mAP): Provides a complete evaluation, considering the average accuracy for all object classes in the dataset. It represents the average performance of the model for all classes. The mAP is calculated by averaging the accuracy values obtained for each object class. The simplified mAP formula is as follows:

$$mAP = (AP(class1) + AP(class2) + ... + AP(classN))/N \quad (4)$$

Where N is the total number of object classes, and AP(classi) is the average accuracy for class i.

A high mAP value indicates that the model performs on average for all object classes. This measure is often used to compare the performance of different object detection models.

2) Frame rate: Frames per second (FPS) is a critical metric for assessing the performance of real-time systems, especially in computer vision and video processing applications. FPS quantifies the number of frames that a system can process within one second, directly impacting the system's perceived fluidity and responsiveness. High FPS values indicate effective processing capabilities essential for object detection, tracking, and recognition tasks in dynamic environments. Maintaining a high FPS while ensuring accuracy presents a significant challenge when deploying deep learning models on resource-constrained architectures. This necessitates careful optimization of the model and hardware to balance computational load and processing speed. Evaluating FPS alongside metrics, such as mAP, offers a comprehensive view of system performance, confirming that it meets practical applications' real-time requirements [26].

In our work, achieving high detection accuracy ensures safety and reliability in real-world scenarios. For object detection tasks in advanced driver assistance systems (ADAS), such as identifying cars, pedestrians, and bicycles, the minimum acceptable mean average precision must exceed 70-75% to guarantee dependable performance. Furthermore, real-time performance should be maintained, typically around 30 FPS, to meet the processing demands of ADAS applications [27]. This requirement requires thoroughly optimizing the YOLOv4 model and the underlying hardware, such as the Jetson Nano, to manage the computational load while preserving high detection accuracy effectively.

IV. Model Optimization

YOLOv4 has garnered significant recognition for its high performance in object detection, showcasing an impressive ability to generalize across diverse datasets. However, the inherent computational complexity of the model poses a substantial challenge when it comes to deploying it in real-time on low-power embedded devices, particularly in edge computing scenarios.

In our research, we trained the YOLOv4 model using the Darknet framework on a high-performance workstation equipped with a robust GPU to manage the extensive computations necessary for training. We focused on a targeted subset of the KITTI dataset, which is well-known for representing real-world driving scenarios. The training emphasized three relevant object classes: Cars, Pedestrians, and Cyclists, enhancing the model's proficiency in detecting these critical elements within urban environments. To optimize the model for real-time applications, all input images used during training were systematically resized to a standardized resolution of 416×416 pixels. This resolution was thoughtfully chosen as it effectively balances robust detection accuracy and efficient processing speed, making it well-suited for applications that require timely responses.

To assess the performance of the trained model, we conducted inference on two architectures: the workstation utilized for training and NVIDIA's Jetson Nano, a low-power embedded architecture. This cross-evaluation allowed us to compare the model's performance in an unconstrained environment (the workstation) with that in a resource-constrained setting (the Jetson Nano). We aimed to analyze the model's fundamental performance before any optimization efforts. The inference results indicate that although the model demonstrates robust accuracy and speed on the workstation, the Jetson Nano experiences a significantly lower inference frequency due to its constrained hardware resources, including GPU and memory. A detailed comparison of YOLOv4 performance across both architectures is summarized in Table I.

TABLE I. YOLOv4 Performance Comparison

Metric	Workstation	Jetson AGX	Jetson Nano
mAP (%)	92.86	89.16	84.25
FPS	91.7	11	< 1

The results show that YOLOv4 achieves a detection accuracy of 92.86% mAP on a workstation but drops to 89.16%, 84.25% mAP on the Jetson AGX and Nano, respectively, due to their lower memory bandwidth and GPU power. Regarding inference speed, the workstation processes images at 91.7 FPS, enabling real-time detection, while the Jetson Nano and AGX perform significantly slower, below the threshold for real-time applications. This highlights that although YOLOv4 is accurate, it demands high computational resources, making it less suitable for low-power embedded devices without further optimization. In this regard, quantization presents itself as a viable solution to reduce latency and resource consumption while still preserving acceptable levels of accuracy. This approach will be elaborated upon in the following subsection.

A. Quantization

Quantization is a model compression technique that converts floating-point numbers (FP32) to lower-bit representa-

tions (such as FP16 or INT8), reducing memory usage, increasing inference speed, and minimizing power consumption for deployment on edge devices. It can be achieved through post-training quantization (PTQ) and quantization-aware training (QAT).

1) Post-Training Quantization (PTQ): is implemented after the training phase and is designed to enhance a neural network model's memory and computational efficiency without significantly diminishing accuracy [28]. This technique is particularly beneficial for well-trained models that require adaptation for deployment in environments with limited resources.

2) Quantization-Aware Training (QAT): in contrast, integrates the quantization process directly into the training phase of the model. This methodology enables the model to acclimate to the effects of quantization throughout the training period, leading to enhanced performance and accuracy when functioning with reduced precision, as opposed to Post-Training Quantization (PTQ) [29].

Post-training quantization is frequently favored for edge deployment scenarios because of its simplicity and efficiency, as it eliminates the need for model retraining. In this work, we utilized the PTQ approach to quantize our YOLOv4 model, which allows for reduced accuracy during inference while enhancing runtime performance without modifying the original learning process.

B. Conversion Pipeline

The YOLOv4 model was initially trained using the Darknet framework, necessitating its conversion into a format compatible with NVIDIA TensorRT. This optimized inference engine facilitates hardware acceleration and precision reduction, which is crucial for efficient deployment on embedded devices. To achieve this, we developed a three-stage conversion and optimization pipeline that transforms the native model into a highly optimized inference engine.

The steps of the pipeline are as follows:

1) Export to ONNX: The first step involved exporting the trained YOLOv4 model in ONNX (Open Neural Network Exchange) format [30]. This intermediate format ensures interoperability among various deep learning frameworks and facilitates subsequent optimization using NVIDIA tools.

2) Model verification and optimization: The exported ONNX model was verified to confirm its structural and functional compatibility with TensorRT. This verification process included validating the network layers, identifying unsupported operations, and making necessary adjustments.

3) Quantization and engine generation with TensorRT: After the ONNX model was validated, we employed TensorRT to perform post-training quantization (PTQ) and generate two optimized versions of the model: one in FP16 (half-precision) and the other in INT8 (8-bit integers). The FP16 model significantly reduces memory consumption and accelerates inference, while the INT8 model drastically minimizes computational requirements. For INT8 quantization, a calibration dataset was utilized to estimate the scaling factors and zero points essential for converting weights and activations while preserving model accuracy.

The conversion and quantization process is illustrated in Fig. 2.

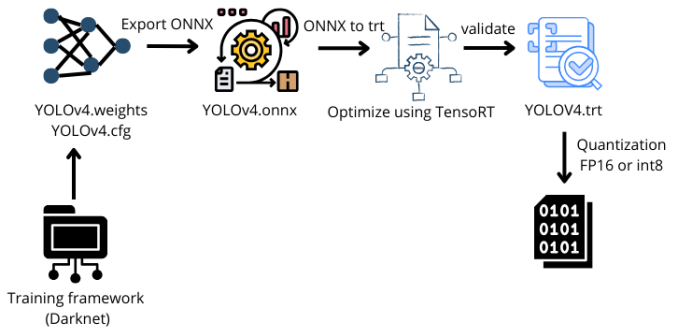

Fig. 2. YOLOV4 quantization and deployment pipeline.

This dual quantization approach allowed us to develop two optimized inference engines tailored for on-board deployment on the Jetson Nano. The models quantized in FP16 and INT8 demonstrated substantial enhancements in inference speed and memory efficiency. A comprehensive performance analysis, encompassing both quantitative and qualitative results, is provided in the following section.

V. Results and Discussion

In this section, we present and analyze the experimental results obtained from evaluating the YOLOv4 model before and after quantization across two hardware architectures: a high-performance workstation, NVIDIA Jetson AGX Xavier, and Jetson Nano. Our analysis focuses on the trade-offs between speed, model size, and precision across the FP32, FP16, and INT8 precision modes.

A. Quantitative Performance

1) Precision and speed analysis: The FP32 version of the YOLOv4 model was evaluated on a high-performance workstation, Jetson AGX, and Nano to establish a baseline for accuracy and inference speed. As indicated in Table II, the model achieved an average mean Average Precision (mAP) of 72.39% on the workstation, displaying commendable performance across various classes: 71.43% for pedestrians, 77.42% for cyclists, and 68.31% for cars. On the Jetson AGX, the FP32 model achieved a mean Average Precision (mAP) of 71.36%, only marginally below the workstation. The per-class AP was also consistent, recording values of 83.24% for cars, 74.56% for pedestrians, and 56.3% for cyclists. These results affirm that the AGX platform, despite being an embedded system, delivers inference performance that closely rivals that of a desktop workstation regarding detection accuracy. This remarkable performance can be attributed to AGX's robust GPU architecture and deep integration with NVIDIA's TensorRT engine, enabling efficient processing of high-precision models while striking a balance between computational throughput and accuracy. Similarly, on Jetson Nano, the overall mAP experienced a slight decline to 68.5%. Notably, the Car class exhibited an improved mAP of 80%, likely due to enhanced detection stability for larger objects on the embedded hardware, possibly due to simplified scene complexity or advantageous scaling of the input image.

The distinction between these architectures was particularly evident in their inference speeds. The model achieved an impressive 140.9 FPS on the workstation, while the Jetson AGX yielded a lower yet commendable 16 FPS. This outcome demonstrates the AGX's superior computing capabilities compared to the Nano but also highlights the challenges of running full-precision models on embedded architectures. In contrast, the Jetson Nano struggled to surpass 1 FPS, emphasizing the significant impact that resource limitations have on inference performance. These results indicate that the original YOLOv4 architecture in its FP32 form is unsuitable for real-time deployment on edge devices like the Nano and, to a lesser degree, the AGX without implementing additional optimization strategies such as quantization or model compression.

After implementing TensorRT-based quantization to FP16, the model recorded an mAP of 48.07% on the workstation, reflecting a significant decrease in detection accuracy compared to the original FP32 version. The average precision (AP) results for each class were 46.60% for cars, 57.78% for cyclists, and 39.84% for pedestrians. This decline in performance can be linked to the sensitivity of quantization in particular layers, especially those associated with smaller object classes like pedestrians.

Despite the reduced accuracy observed on the workstation, deploying the FP16 model on the Jetson AGX and Jetson Nano yielded promising results. On the Jetson AGX, the mean Average Precision (mAP) reached 68.33%, with Average Precision (AP) scores of 80.23% for cars, 71.32% for pedestrians, and 53.45% for cyclists. In addition to its accuracy, the Jetson AGX achieved an inference speed of 47.6 FPS, showcasing its ability to leverage hardware-accelerated FP16 operations effectively. The Jetson Nano also performed admirably, maintaining an mAP of approximately 68.62%, nearly identical to the unoptimized FP32 model, while significantly improving its inference speed to 3 FPS. These results indicate that TensorRT managed quantization-induced errors well across both embedded architectures. Consequently, the FP16 variant emerges as a strong candidate for applications requiring a balanced approach between accuracy and computational efficiency.

To further enhance inference efficiency and reduce resource consumption, additional quantization to INT8 precision was implemented. On the workstation, the INT8 model achieved an mAP of 38.96%, with AP scores of 36.63% for cars, 40.82% for cyclists, and 39.42% for pedestrians. As expected, the performance degradation was more pronounced than with the FP16 model due to the greater quantization error associated with lower bit precision. Nonetheless, the INT8 model demonstrated exceptional runtime performance on both embedded architectures. On the Jetson AGX, the model maintained a respectable mAP of 56.69% with an inference speed of 62.5 FPS, outperforming the Jetson Nano in terms of throughput. Similarly, on the Jetson Nano, the mAP remained stable at 68.5%, while inference speed reached 5 FPS, marking the highest observed among all tested variants. This consistency across the quantized models on the Nano suggests that the quantization process was effectively calibrated and further highlights the robustness of the YOLOv4 architecture when utilized on edge platforms employing lower-precision computation.

2) Model Size Analysis: The graph below Fig. 3 illustrates the storage sizes of the YOLOv4 model across three precision formats (FP32, FP16, INT8) on the Jeson AGX, Jetson Nano and Workstation. The findings indicate that quantization significantly reduces model size, with the INT8 version being the most compact, followed by FP16, while FP32 remains the largest. On the Jetson Nano, the model size decreases from 586.5 MB in FP32 to 201.6 MB in INT8, representing a decline of nearly 66%. Similarly, on the workstation, the size is reduced from 400.3 MB to 69.1 MB, an impressive reduction of over 82%. On the Jetson AGX, the model size decreases from 592.7 MB in FP32 to 77.8 MB in INT8, resulting in a significant relative reduction of approximately 86.9% the most substantial among all three platforms. This indicates that the quantization pipeline on the Jetson AGX may be more effectively optimized or better integrated with the TensorRT engine, leading to more efficient memory utilization. This substantial decrease is significant for edge deployment, where constraints on memory and loading times can directly impact real-time performance. Additionally, smaller models consume less power and provide faster start-up and inference times, making the INT8 format a compelling choice for resource-constrained environments.

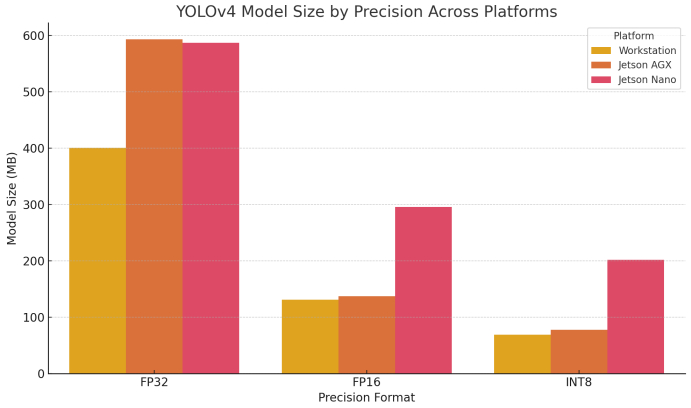

Fig. 3. Model size comparison.

Despite employing identical model architectures and quantization techniques, the model size on the workstation consistently remains smaller than that on the Jetson Nano, irrespective of the accuracy level achieved. This disparity primarily arises from how TensorRT compiles models within architecture-specific inference engines.

The performance improvements achieved through quantization are primarily due to decreased computational costs and better hardware utilization. By transforming FP32 operations into FP16 and INT8 formats, the model takes advantage of faster arithmetic and lower memory usage, which is particularly beneficial for embedded GPUs. Furthermore, the TensorRT produces an optimized binary encompassing model weight alongside device-specific execution plans, kernel selections, memory layouts, and fallback mechanisms. The binary for the Jetson Nano typically includes more metadata and execution pathways to ensure compatibility across various components (such as the GPU, DLA, and CPU), contributing to a larger binary size. In contrast, the workstation engine benefits from a robust and stable GPU environment, allowing TensorRT

TABLE II. SUMMARY OF YOLOV4 PERFORMANCE BEFORE AND AFTER QUANTIZATION

Model	Architecture	mAP (%)	AP_{Car}	$AP_{Pedestrian}$	$AP_{Cyclist}$	FPS
FP32	Workstation	85.68	91.30	87.50	78.22	140.93
	Jetson AGX	71.36	83.24	74.56	56.3	16
	Jetson Nano	68.68	80.22	71.50	54.30	< 2
FP16	Workstation	48.07	46.60	57.78	39.84	297.56
	Jetson AGX	68.33	80.23	71.32	53.45	47.6
	Jetson Nano	68.62	80.18	71.50	54.18	3
INT8	Workstation	38.96	36.63	40.82	39.42	372.97
	Jetson AGX	56.69	72.28	53.64	45.03	62.5
	Jetson Nano	68.5	80	71.5	54.1	5

to optimize the model more aggressively and eliminate certain fallback functions.

Although post-training quantization significantly enhances inference efficiency, our findings indicate that achieving real-time performance on the Jetson Nano remains elusive, even with INT8 quantization. While a five-fold speedup has been realized, Nano's limited computational resources constrain its ability to fully leverage the advantages of low-precision inference, culminating in a maximum performance of only 5 FPS, which does not meet real-time requirements. In contrast, the Jetson AGX Xavier, equipped with a more powerful GPU and dedicated inference accelerators, successfully achieves real-time processing at 48 FPS using FP16 quantization. This disparity underscores that as hardware resources diminish, the capacity to attain real-time object detection declines, irrespective of software-level optimization.

Furthermore, the relatively stable accuracy observed on the Nano across FP32, FP16, and INT8 quantization levels can be attributed to its inability to fully quantify all model components due to hardware limitations and calibration challenges. Consequently, some layers may revert to higher-precision computation (e.g. FP16), which helps maintain detection accuracy but restricts the performance improvements typically expected from INT8 execution. These findings emphasize the need to align quantization strategies with the capabilities of the target hardware in order to strike a balance between efficiency and precision.

Compared to the work by [11], which attained high accuracy using YOLOv7 on the Jetson Nano, our study investigates the deployment of YOLOv4 on both the Jetson Nano and AGX Xavier through multi-level quantization. While [11] concentrated on optimizing accuracy, our findings underscore the balance between speed and precision, revealing that only the AGX with FP16 quantization achieves real-time performance. This underscores the significance of aligning quantization strategies with hardware capabilities for effective embedded deployment.

B. Qualitative Performance

To complement the quantitative evaluation, a qualitative analysis was conducted to visually assess the detection performance of the YOLOv4 model when deployed on the Jetson Nano using the quantified INT8 version. This subsection presents sample outputs that illustrate the model's capability to identify and classify objects in real-world scenarios. Selected images demonstrate how the INT8 model operates under resource constraints, particularly regarding bounding box accuracy and consistency of class labels. These visual results offer practical insights into the model's effectiveness following quantization and underscore any noticeable degradation in detection quality resulting from the compression process.

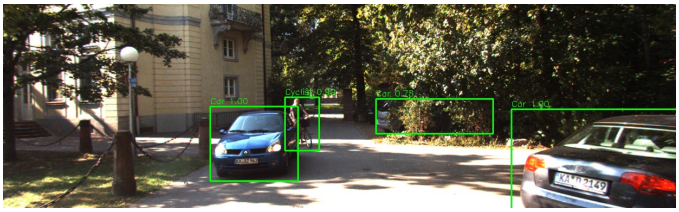

Fig. 4. Car detection.

Fig. 5. Pedestrian detection.

Fig. 6. Cyclist detection.

The visual results in Fig. 4 to 6 showcase the YOLOv4 model's performance using INT8 on the Jetson Nano in various urban traffic scenarios. Despite resource constraints, the model delivers reliable detection for cars, pedestrians, and cyclists. In Fig. 4, the model accurately identifies several vehicles and in a moderately congested area, with confidence scores between 0.78 and 1.00. It effectively distinguishes vehicles, highlighting its ability to handle overlapping classes. Fig. 5 focuses on detection in a semi-congested alleyway, recognizing six pedestrians and one cyclist, all with confidence scores

above 0.81. The model excels even with partial occlusions, reaffirming high average pedestrian precision from previous evaluations. In Fig. 6, the model excels in an open road scenario, confidently detecting a nearby cyclist (1.00) and a distant vehicle (0.99).

Overall, these qualitative results demonstrate the effectiveness of INT8 quantization, confirming that YOLOv4 with TensorRT is well-suited for embedded vision applications, maintaining strong localization and accurate classifications.

VI. Conclusion

In this study, we examined the deployment and optimization of the YOLOv4 object detection model on embedded platforms, specifically focusing on the NVIDIA Jetson Nano and Jetson AGX Xavier. We began with an FP32 model trained in Darknet and subsequently applied TensorRT-based post-training quantization (FP16 and INT8) to enhance operational efficiency.

The quantization process significantly accelerated inference speed and reduced the model size with minimal impact on accuracy. On the Jetson Nano, the INT8 model achieved a five-fold increase in speed and a 65% reduction in size while maintaining a consistent mean Average Precision (mAP). On the Jetson AGX, the FP16 and INT8 models reached speeds of up to 48 and 62 FPS, respectively, demonstrating nearly real-time performance with high accuracy retention.

Future work will integrate quantization-aware training (QAT) to reduce accuracy loss when using INT8 precision. Furthermore, deploying mixed-precision and runtime-adaptive quantization strategies, along with an extension to FPGA hardware and incorporating more diverse datasets, will strengthen the robustness and generalizability of the optimized models for real-world embedded applications.

Acknowledgment

This work was partially funded by the Ministry of Europe and Foreign Affairs, (Eiffel grant number: 116724T), and by the National Center for Scientific and Technical Research of Morocco (Grant number: 30UM5R2021).

References

[1] P. Jha, D. Dembla, and W. Dubey, "Implementation of machine learning classification algorithm based on ensemble learning for detection of vegetable crops disease." *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 1, 2024.

[2] X. Jihong, Z. Xiang *et al.*, "Edge computing for real-time decision making in autonomous driving: Review of challenges, solutions, and future trends." *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 7, 2024.

[3] H. Sun, S. Zhang, X. Tian, and Y. Zou, "Pruning detr: Efficient end-to-end object detection with sparse structured pruning," *Signal, Image and Video Processing*, vol. 18, no. 1, pp. 129–135, 2024.

[4] K. Acharya, A. Velasquez, and H. H. Song, "A survey on symbolic knowledge distillation of large language models," *IEEE Transactions on Artificial Intelligence*, 2024.

[5] D. Zhang and Y. Chen, "Lightweight fire detection algorithm based on improved yolov5." *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 6, 2024.

[6] Q. Li and S. Duan, "Road surface crack detection based on improved yolov9 image processing." *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 11, 2024.

[7] H. M. Chiam, Y. C. Wong, R. S. S. Singh, and T. J. S. Anand, "Energy optimized yolo: Quantized inference for real-time edge ai object detection," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 17, no. 1, pp. 19–28, 2025.

[8] H. Lokhande and S. R. Ganorkar, "Object detection in video surveillance using mobilenetv2 on resource-constrained low-power edge devices," *Bulletin of Electrical Engineering and Informatics*, vol. 14, no. 1, pp. 357–365, 2025.

[9] K. Abdulhaq and A. A. Ahmed, "Real-time object detection and recognition in embedded systems using open-source computer vision frameworks," *Int. J. Electr. Eng. and Sustain.*, pp. 103–118, 2025.

[10] J. N. Chaudhari, H. Galiyawala, P. Sharma, P. Shukla, and M. S. Raval, "Onboard person retrieval system with model compression: A case study on nvidia jetson orin agx," *IEEE Access*, 2025.

[11] S. Shekhar, T. Sathwik, M. Pritwani, R. Kumar, K. Sreelakshmi *et al.*, "Advancing deep learning on edge devices: Fine-tuning and deployment of yolov7 model for efficient object detection in ai based computer vision applications," in *2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*. IEEE, 2025, pp. 1912–1918.

[12] P. Ruiz-Barroso, F. M. Castro, and N. Guil, "Real-time unsupervised video object detection on the edge," *Future Generation Computer Systems*, p. 107737, 2025.

[13] A. Zagitov, E. Chebotareva, A. Toschev, and E. Magid, "Comparative analysis of neural network models performance on low-power devices for a real-time object detection task," *Computer Optics*, vol. 48, no. 2, pp. 242–252, 2024.

[14] C. Cheng, "Real-time mask detection based on ssd-mobilenetv2," in *2022 IEEE 5th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*. IEEE, 2022, pp. 761–767.

[15] K. Zhao and W. Q. Yan, "Fruit detection from digital images using centernet," in *Geometry and Vision: First International Symposium, ISGV 2021, Auckland, New Zealand, January 28-29, 2021, Revised Selected Papers 1*. Springer, 2021, pp. 313–326.

[16] L. Ma, L. Zhao, Z. Wang, J. Zhang, and G. Chen, "Detection and counting of small target apples under complicated environments by using improved yolov7-tiny," *Agronomy*, vol. 13, no. 5, p. 1419, 2023.

[17] S. Ali, A. Siddique, H. Ates, and B. Gunturk, "Improved yolov4 for aerial object detection," in *Signal Processing and Communications Applications Conference*, 2021.

[18] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[19] W. Chen, J. Luo, F. Zhang, and Z. Tian, "A review of object detection: Datasets, performance evaluation, architecture, applications and current trends," *Multimedia Tools and Applications*, vol. 83, no. 24, pp. 65 603–65 661, 2024.

[20] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[21] Y. Cui, S. Lu, and S. Liu, "Real-time detection of wood defects based on spp-improved yolo algorithm," *Multimedia Tools and Applications*, vol. 82, no. 14, pp. 21 031–21 044, 2023.

[22] F. Z. Guerrouj, S. Rodríguez Flórez, M. Abouzahir, A. El Ouardi, and M. Ramzi, "Efficient gemm implementation for vision-based object detection in autonomous driving applications," *Journal of Low Power Electronics and Applications*, vol. 13, no. 2, p. 40, 2023.

[23] L. Shen, H. Tao, Y. Ni, Y. Wang, and V. Stojanovic, "Improved yolov3 model with feature map cropping for multi-scale road object detection," *Measurement Science and Technology*, vol. 34, no. 4, p. 045406, 2023.

[24] P. Fränti and R. Mariescu-Istodor, "Soft precision and recall," *Pattern Recognition Letters*, 2023.

[25] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, no. 1, p. 6086, 2024.

[26] Y. Lin, "Lightweight ca-yolov7-based badminton stroke recognition: A real-time and accurate behavior analysis method." *International Journal of Advanced Computer Science & Applications*, vol. 16, no. 2, 2025.

[27] A. H. Khan, S. T. R. Rizvi, and A. Dengel, "Real-time traffic object detection for autonomous driving," *arXiv preprint arXiv:2402.00128*, 2024.

[28] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-training quantization on diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 1972–1981.

[29] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," *arXiv preprint arXiv:2008.05000*, 2020.

[30] D. Chang, J. Lee, and J. Heo, "Lightweight of onnx using quantization-based model compression," *The Journal of The Institute of Internet, Broadcasting and Communication*, vol. 21, no. 1, pp. 93–98, 2021.