A Quantized Deep Learning Model for Efficient Plant Leaf Disease Detection on Embedded Systems

Balkis Tej[1], Soulef Bouaafia[2a,b], Mohamed Ali Hajjaji[3], Abdellatif Mtibaa[4], Mohamed Atri[5]
Automatic Signal and Image Processing, Research Laboratory (LR13ES13),
National Engineering School of Monastir, University of Monastir, Monastir, Tunisia[1]
Laboratory of Condensed Matter and Nanosciences (LMCN), Faculty of Sciences of Monastir,
University of Monastir, Tunisia[2a]
Higher Institute of Applied Sciences and Technology of Kairouan, University of Kairouan, Tunisia[2b]
Research Laboratory in Algebra Numbers theory and Intelligent Systems (RLANTIS), University of Monastir, Tunisia[3]
Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse,Tunisia[3]
Systems Integration and Emerging Energies Laboratory (LR21ES14),
National Engineering School of Sfax, University of Sfax, Tunisia[4]
Computer Engineering Department, King Khalid University, Abha, Saudi Arabia[5]

***Abstract*—You Only Look Once (YOLO) object detection network has gained significant adoption in the field of plant leaf disease detection due to its strong detection capabilities. However, deploying YOLO models on resource-constrained devices remains challenging, as they require substantial computational power. The complexity and size of these models pose significant obstacles for edge platforms, which are often limited in processing and memory resources. To address these limitations and accelerate inference, we propose a quantized version of YOLOv5x, called Quant-YOLOv5x. This quantization reduces the size and complexity of the model, making it more suitable for edge deployment while maintaining competitive detection accuracy. The experiments were carried out using a self-generated dataset focused on detecting tomato and pepper leaf diseases. Our quantization method reduces the bitwidth of the entire YOLO network to 8 bits, resulting in only a 2.8% decrease in mean Average Precision (mAP), a 50% reduction in model size, and an increase of 4.7 FPS compared to the standard YOLOv5 model.**

***Keywords*—Object detection; plant disease; quantization technique; YOLOv5**

I. Introduction

In Tunisia, the agricultural sector plays a vital role in both the economy and the sociopolitical landscape, significantly contributing to national objectives such as food security, income generation, employment, and sustainable resource management. Among key crops, tomatoes and peppers are of particular importance. In 2023, Tunisia produced 1.09 million tons of tomatoes [1], positioning the country as one of the leading tomato producers in North Africa. Pepper cultivation ranks third in vegetable agriculture, covering an area of 20,000 hectares with an average annual production of 346,000 tons over the past five years [2]. However, the sector faces significant challenges, particularly the prevalence of plant diseases. In 2021, plant diseases reduced the cultivated area of tomatoes by 15% [3]. Similarly, pepper crops experienced more than a 50% loss of potential production during the peak season due to disease outbreaks [4].

To address these challenges, early and accurate detection of plant diseases is essential. Recent advancements in artificial intelligence, particularly deep learning, have enabled the development of intelligent systems capable of automatically diagnosing plant diseases. Among these techniques, object detection methods have become pivotal for not only identifying but also localizing diseased regions on plant leaves. These methods provide valuable spatial information that enhances the precision of disease management strategies. Convolutional Neural Networks (CNNs) are widely used in this context, offering significant improvements in detection accuracy. Among the various object detection methods, YOLO (You Only Look Once) has attracted significant attention. YOLO has evolved through several versions, from its initial release in 2016 to YOLOv10 in 2024. Our work focuses on YOLOv5, as it has been the subject of more extensive research and well-established studies. However, this investigation can be extended to include newer versions of YOLO in future work.

In 2020, Ultralytics introduced YOLOv5, available in five different model sizes, ranging from nano to extra-large based on the width and depth multipliers [5]. YOLOv5 incorporated several major improvements over previous versions, including updates to the backbone architecture and the introduction of automated hyper-parameter optimization. The backbone features the CSPDarknet53 structure [6], which builds upon Darknet53 with the addition of the Cross Stage Partial (CSP) strategy. YOLOv5's neck leverages CSP-PAN [7], along with an enhanced version of the Spatial Pyramid Pooling block (SPPF), which speeds up processing. For output generation, the head still employs the structure used in YOLOv3. The architecture of YOLOv5 large (YOLOv5l) is shown in Fig. 1, where CSPDarknet53 includes C3 blocks, which are CSP-enhanced modules. The CSP strategy divides the feature map from the base layer into two parts, which are later merged using a cross-stage process. This enables the C3 module to better manage redundant gradients and improve the flow of information between residual and dense blocks. The C3 block is a simplified form of BottleNeckCSP, currently used in the latest versions of YOLOv5. These updates have enabled YOLOv5 to reach state-of-the-art performance on several benchmarks.

The growing trend of using overparameterized models has resulted in improved accuracy, but it has also significantly

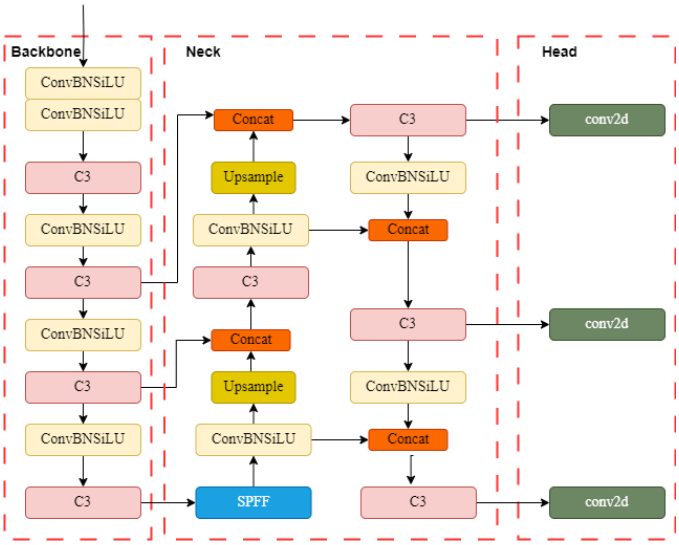

Fig. 1. Standard YOLOv5 architecture.

increased the number of floating point operations (FLOPs) and model parameters required [8]. These requirements hinder the implementation of complicated models on edge platforms due to constraints in memory, power, and processing capacity. To handle this challenge, different techniques have been developed to compress deep learning models to enable the deployment of large models on edge platforms. Model quantization has emerged as a prevalent method for optimizing performance. Within the various methods, model quantization has emerged as a prevalent method to optimize performance.

The main contributions of this paper are summarized as follows: The primary contribution is the development of a hardware-efficient quantized YOLOv5 architecture, called Quant-YOLOv5, which employs a low-bitwidth quantization method to minimize model size, computational complexity, and inference delay for implementation on resource-limited devices to detect plant leaf diseases. This novel architecture includes the simplification of the activation function by replacing the SiLU function with the hardware-friendly ReLU activation function, the combination of Batch Normalization with adjacent convolution layers to reduce computational overhead, and the replacement of the CBS, C3, and SPPF layers with their quantized versions, which use 8-bit integer (Int8) weights and activations instead of floating-point (FP32) values. Additionally, the quantization technique leverages the PyTorch framework and the Xilinx Brevitas library, specifically designed for FPGA deployment, and supports the Quantization-Aware Training (QAT) method. The performance evaluation of the proposed quantized architecture is conducted using a real-world dataset for tomato and pepper leaf diseases, and we compare the proposed approach with state-of-the-art models based on standard performance metrics.

The remaining parts of the paper are organized as follows: Section II presents an overview of the different quantization techniques. Section III describes the proposed methodology. Section IV provides the experimental results and discussion. At last, Section V summarizes the study and outlines potential areas for research.

II. Quantization Techniques Overview

Quantization is a technique that maps a broad range of high-precision values to a reduced collection of discrete values. In terms of CNN models, quantization involves converting weights and activations from floating-point representations to low-bit numbers to decrease memory requirements and computational complexity [9], [10].

Consider a neural layer, denoted by C, with an activation function σ. The function C can be defined as follows:

$$C : \mathcal{E}_{d_e} \to \mathcal{B}_{d_s}, \quad X \mapsto \sigma(WX + B) \tag{1}$$

where d_e and d_s represent the input and output dimensions, and E and B each define a vector space over R. For a fully connected layer, $\mathcal{E} = R^{d_e}$, and for a convolutional layer acting on a 640x640 pixel image, $\mathcal{E} = R^{640\times640}$. The elements W and B are, respectively, the weight tensor and the bias vector associated with the layer.

The main objective is to optimize the computationally intensive operations, particularly the matrix multiplication of $W \times X$. In the quantization process, performance improvement is achieved by replacing traditional 32-bit floating-point arithmetic operations, commonly used in training artificial neural networks, with operations using fixed-point numbers with n bits. Typically, n can take values of 1, 2, 4, 8, or 16.To adapt the elements used in computing the function N to an n-bit representation, a quantization operator, denoted Q, is required. This operator transforms the values of X so that they are expressed within the interval $\{-2^{n-1}-1, \ldots, 2^{n-1}-1\}$. We assume that the range of X is symmetric for simplicity (generally, $\alpha = \max|X|$ is used to define the limits). Thus, the quantization of X is given by:

$$Q(X) \in \{-2^{n-1}-1, \ldots, 2^{n-1}-1\} \quad \text{and} \quad X \in [-\alpha, \alpha] \tag{2}$$

where α represents the maximum absolute value of X. Consider a basic quantization operator, defined as:

$$Q : X \mapsto \left[X \times \frac{2^{n-1}-1}{\alpha} \right] \in \{-2^{n-1}-1, \ldots, 2^{n-1}-1\} \tag{3}$$

where $[\cdot]$ denotes rounding to the nearest integer. In this general configuration, the function equation for the layer becomes:

$$C : \mathcal{E}_{d_e} \to \mathcal{B}_{d_s}, \quad X \mapsto \sigma\left(Q(W)Q(X) + B\right) \tag{4}$$

It is important to note that the bias B is not subject to quantization. Typically, in inference algorithms, the bias is handled independently by the system performing the computations on a specific hardware architecture. A major concern is determining whether quantization may introduce a significant error. Indeed, if the distribution of X greatly differs from the interval $\{-2^{n-1}-1, \ldots, 2^{n-1}-1\}$, initial quantization can significantly alter the data scale. For this reason, it is necessary

to implement a re-normalization step, or "dequantization." This operation is defined as:

$$Q^{-1} \circ Q : X \mapsto \alpha \frac{X}{2^{n-1}-1} \left[X \times \frac{2^{n-1}-1}{\alpha} \right] \in [-\alpha, \alpha] \quad (5)$$

Consequently, the function for the layer becomes:

$$C : \mathcal{E}_{d_e} \to \mathcal{B}_{d_s}, \quad X \mapsto \sigma \left(Q^{-1}(Q(W)Q(X)) + B \right) \quad (6)$$

The error due to quantization is thus a rounding error relative to the input scale. We can explicitly express this layer function as follows:

$$C(X) = \sigma \Bigg(\left(\frac{\max|X|}{2^{n-1}-1} \frac{\max|W|}{2^{n-1}-1} \right) \times \left[W \times \frac{2^{n-1}-1}{\max|W|} \right] \times \left[X \times \frac{2^{n-1}-1}{\max|X|} \right] + B \Bigg) \quad (7)$$

An examination of this formulation reveals several important observations. First, it should be noted that the number of bits allocated for the quantization of inputs X and weights W does not necessarily have to be identical. It is common in the literature to designate by $W8/A4$ the practice of quantizing weights to 8 bits and activations to 4 bits.

Next, we observe that the scaling factor $\lambda_X = \frac{\max|X|}{2^{n-1}-1}$, used for inputs, is also applied to the outputs, implying it should have a dimension of 1. On the other hand, the scaling factor $\lambda_Y = \frac{\max|Y|}{2^{n-1}-1}$, used for weights, can take the form of a vector with d_s dimensions, introducing the concept of per-channel quantization. Finally, it is essential to note that if $Q(W)$ and $Q(X)$ are quantized on n bits, their product could exceed this limit, leading to overflow (modulo). To prevent this risk, intermediate calculations are often performed on 32 bits, a technique known as the 32-bit accumulator. In modern inference systems, de-quantization (Q^{-1}) and quantization (Q) operations are seamlessly integrated to avoid costly conversions from integers to floating-point numbers. Another notable benefit of quantization, in addition to accelerating inference, is the reduction in memory consumption. A large portion of the memory cost in neural networks is attributed to storing weights. Quantization not only reduces the data size for these weights but also increases redundancy, which enhances the efficiency of compression techniques without loss. Algorithm 1 and Fig. 2 summarize the quantization principles discussed previously, illustrating how weights and activations are adapted for increased efficiency in terms of both memory usage and computational speed.

Algorithm 1 Quantization Algorithm

Input: Weights W, activations X, number of bits b for quantization

Output: Quantized weights $Q(W)$, quantized activations $Q(X)$

Function `Quantize(`W, X, b`)`**:**

```
// Calculate the absolute maximum for weights and activations
```
$\alpha_W \leftarrow \max(|W_i|)$; $\alpha_X \leftarrow \max(|X_i|)$;
```
// Quantization of weights and activations
```
$Q(W) \leftarrow \left\lfloor W \times \frac{2^{b-1}-1}{\alpha_W} \right\rfloor$; $Q(X) \leftarrow \left\lfloor X \times \frac{2^{b-1}-1}{\alpha_X} \right\rfloor$;
```
// De-quantization of weights to maintain scale
```
$W' \leftarrow Q(W) \times \frac{\alpha_W}{2^{b-1}-1}$;

return $Q(W), Q(X)$;

Quantization methods are generally divided into two categories: Quantization Aware Training (QAT) and Post Training Quantization (PTQ).

A. Post Training Quantization (PTQ)

Let us assume the existence of a pre-trained neural network R. The quantization operator Q mentioned earlier can be directly applied to the weights of this network. However, for activations of intermediate layers, such direct application is not possible without knowing the value of α (i.e., the maximum absolute value of $|X|$). To address this lack of information, Nagel et al. [11] suggested using accumulated statistics in the batch normalization layers make it possible to perform quantization after training. We denote by BN a batch normalization layer placed just before the layer to be quantized. Consider the transformation performed by a batch normalization layer, defined as follows:

$$BN : X \mapsto \gamma \frac{(X-\mu)}{(\alpha+\epsilon)} + \beta \quad (8)$$

where γ and β are learned parameters, μ and α are calculated means and standard deviations, and ϵ is a small number to prevent division by zero. When the batch normalization layer is properly optimized, the expected value of $BN(X)$ is β and its variance is γ^2. Thus, it is possible to select a margin of six standard deviations (although the authors recommend this value, in practice, a range from 5 to 10 is also effective) to ensure that:

$$BN(X) \in [\beta - 6|\gamma|, \beta + 6|\gamma|] \quad (9)$$

This range can then be used to define the limits of the input values for the next layer in the network. This method allows for precise quantization after training by using the statistics of the previous layer's activations. Post-training quantization uses the original pre-trained full-precision model without the need for further training. As a result, it offers a more efficient quantization process, requiring less time and fewer resources than training-dependent quantization methods. Although it doesn't necessitate a complete dataset, post-training quantization methods use calibration techniques to

quantize the output activations. The authors in [12] present application of PTQ technique on an improved YOLOv5-based algorithm for tomato detection and ripeness assessment. Upon training, they quantized the model to 16 bits using the Nihui Convolutional Neural Network (NCNN) framework, achieving a 51.1% reduction in model size, an 18.8 fps increase, with a 0.9% decrease in accuracy. Zhang et al. [13] introduced a post-training quantization (PTQ) framework based on a greedy path-following mechanism to iteratively adjust quantization settings, reducing computational complexity and memory utilization without model retraining. This technique was tested on ImageNet and achieved near-original model accuracy with 5-bit quantization under various bit-width constraints. Banner et al. [14] propose a 4-bit PTQ approach using Analytical Clipping for Integer Quantization (ACIQ), bias correction, and per-channel bit allocation to reduce quantization errors. These methods restore most accuracy loss without training, showing a few percentage decreases compared to full-precision models spanning convolutional neural networks like ResNet and VGG. By incorporating statistical insights and closed-form analytical answers, the method improves low-bit quantization schemes' usability and enables rapid implementation in limited situations. In the study of [15], post-training quantization (PTQ) is used on the LeNet-5 model, leveraging both symmetric and asymmetric quantization techniques to reduce model complexity and size. Evaluated on the MNIST and Fashion-MNIST datasets, the quantized 16-bit models showed the same level of accuracy as the original 32-bit floating-point models. The 8-bit versions, on the other hand, showed small accuracy drops of 0.01% and 0.02%, respectively. The results show that PTQ can reduce model size by up to four times and improve the efficiency of storage and transmission, which means it can be used on devices with limited resources.

B. Quantization Aware Training

The Post Training Quantization (PTQ) technique offers several advantages, notably its simplicity of implementation, as it does not require any quantization during the training phase of a pre-trained network. However, parameters defined from the final floating-point model can exhibit quantization errors. These errors could lead to misclassifications due to the increased quantization error. Quantization during training (QAT) is then used to address these errors by adjusting the parameters during the training phase. Since QAT integrates the quantization process during training and adjusts the parameters accordingly, it provides a notable advantage in improving optimization more significantly [16]. The overall approach relies on using a quantized version of the network for computations during training, both in forward and backward propagation, while maintaining a copy of the parameters in full precision (such as weights and biases). Conventional training techniques for neural networks primarily rely on various forms of gradient descent. However, the quantization operator Q, as defined in Eq. (2), generally has an almost zero gradient everywhere, making it difficult to apply gradient descent directly. To address this problem, researchers have proposed the use of "straight through" (STE) estimators facilitates efficient learning despite these challenges. This approach involves treating the derivative of the de-quantization operator $Q^{-1} \circ Q$ as the identity, i.e.:

$$\nabla(Q^{-1} \circ Q) = Id \tag{10}$$

This allows bypassing the problem of a null gradient. This technique is equivalent to eliminating the rounding step during the optimization process. Intuitively, it involves maintaining a floating-point version of the network. After an initial inference using the quantized network, the floating-point version is used to backpropagate the error. Subsequently, the quantized version of the network is returned for the next phase of training.

In study [17], the authors presented a QAT methodology that adaptively establishes the rounding strategy for weights according to the direction of their updates during training while adjusting the corresponding gradient. Their approach included layer-wise quantization, implementing symmetric clipping ranges for weights and asymmetric ones for activations within the network. Authors in study [18] modify the architecture of YOLOv5 backbone with SuffleNetV2 architecture and decrease the layer depth in the PAN and head networks, optimizing the model for mobile devices. Using QAT techniques with TensorFlow Lite Micro, they quantize weights and activations to 8-bit accuracy, enabling deployment on an ultra-low-power microcontroller from the STM32 family. This optimization reduces the model size by 87.5MB, decreases latency by 92.7, and results in a 14% accuracy loss.

Subsequent works [19] expanded these concepts to higher bit widths and also addressed the quantization of gradient signals. Other methods propose learning parameters that define the range of activation signals within the network [20], as well as new gradient estimations that facilitate the learning of appropriate scaling factors for the integer quantization of weights and activations [21].

Fig. 2 illustrates the fundamental principles of the two quantization methods, QAT and PTQ, used in neural network architectures.

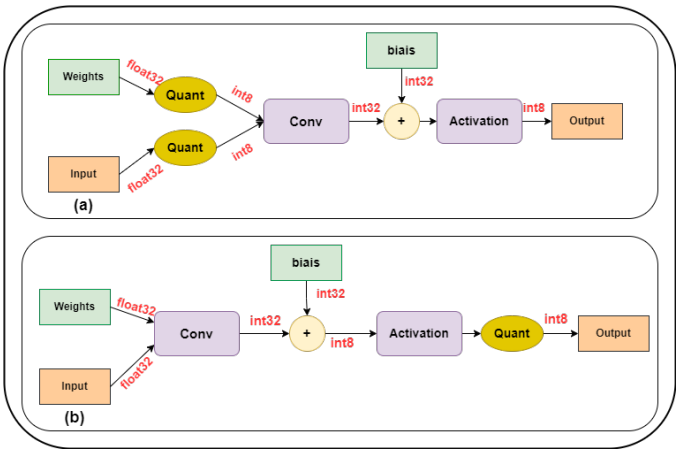

Fig. 2. Diagram quantization processes: (a) Training-aware quantization (QAT), (b) Post-training quantization (PTQ).

III. Proposed Quantized Deep Learning Model: Quantized YOLOv5

A. Proposed Methodology

Although YOLOv5 models show efficacy in the detection of plant diseases, deploying these networks on edge platforms necessitates considerable computational resources, resulting in increased processor consumption and large memory bandwidth consumption. These requirements create significant challenges for edge platforms, which have limited computing and memory capabilities. Field-programmable Gate Arrays (FPGAs) address these difficulties. FPGAs enable accelerated computations while reducing energy consumption, making them well-suited for embedded applications. Despite this solution, the complexity and size of YOLO models remain a barrier for edge platforms. To overcome these challenges, we adopted neural network quantization, which reduces the size of the model while maintaining acceptable performance. As illustrated in Fig. 3, our methodology begins with the development of Quant-YOLOv5, a quantized version of the YOLOv5 model. Tensorflow and Pytorch are the most commonly used frameworks to quantify DL models. These frameworks quantify DL models for resource-constrained CPU or GPU hardware. However, these frameworks do not support FPGA-based hardware accelerators. Brevitas is a Pytorch library for creating QNN models with various extraction levels, which can be deployed to Xilinx FPGAs using the experimental framework FINN [22]. Our work aims to produce deep learning models suitable for devices with constrained computational resources, such as FPGAs. Consequently, we used the Brevitas framework to quantize the YOLOv5 model, as it serves as a single framework that supports the design, development, and execution of quantized neural networks in FPGAs [23], [24], [25]. After training and quantizing the model, it is exported in ONNX format, which ensures portability and facilitates its transformation for hardware-specific applications. Finally, this standardized model is converted into FPGA logic using the FINN framework. Finally, model inference could be performed directly on the FPGA architecture, effectively addressing memory and computational resource constraints while maintaining optimal performance.

B. Proposed Quantized YOLOv5 Architecture

For the development of our Quant-YOLOv5 model, we used Brevitas [26], a PyTorch library developed by Xilinx Research Lab, specifically designed for quantization-aware training (QAT). A unique feature of Brevitas is its capability to quantize input features, an advantage not found in many other frameworks, providing significant resource savings for resource-constrained devices like FPGAs. It enables the transformation of traditional PyTorch layers into their quantized equivalents, creating hybrid models that combine quantized and nonquantized layers. This capability is achieved through Brevitas' interoperability with PyTorch layers, allowing for the selective quantization of critical parts of the network, including weights and activations. Brevitas plays a key role in producing models compatible with FINN, offering a wide range of options for designing quantized neural networks. The type of quantization supported by Brevitas is uniform quantization.

Uniform quantization is based on three key parameters: the scale factor, the zero point, and the bit width. The scale factor and zero point are used to map a floating point value to an integer value, the size of which depends on the bit width. The scale factor is typically represented as a floating point number that determines the quantization step size. The zero point is an integer that ensures the correct quantization of the zero value. This is essential to avoid issues when applying functions such as ReLU or shift operations [27]. Once these parameters are defined, the quantization process can begin. Starting with a real value x, it is transformed into an unsigned integer $\hat{x}$ according to the formula:

$$\hat{x} = \text{clamp}\left(\left\lfloor \frac{x}{s} + z \right\rceil, 0, 2^b - 1\right) \tag{11}$$

where $\lfloor \cdot \rceil$ represents the rounding operator to the nearest integer, and the clamp function is defined as:

$$\text{clamp}(x; a, c) = \begin{cases} a & \text{if } x < a, \\ x & \text{if } a \leq x \leq c, \\ c & \text{if } x > c. \end{cases} \tag{12}$$

To approximate the initial real value, a dequantization step is necessary:

$$\tilde{x} = s(\hat{x} - z) \tag{13}$$

By combining these two steps, we obtain a general definition of the quantization function $q(x)$:

$$\hat{x} = q(x; s, z, b) = \text{clamp}\left(\left\lfloor \frac{x}{s} + z \right\rceil, 0, 2^b - 1\right) \tag{14}$$

With this dequantization step, we can also define the limits of the quantized values ($\hat{x}_{\text{min}}$ and $\hat{x}_{\text{max}}$) as follows: $\hat{x}_{\text{min}} = z$ and $\hat{x}_{\text{max}} = 2^b - 1 + z$. Values x that exceed these limits will then be truncated, resulting in a clipping error. To reduce this error, it is possible to increase the quantization scale. However, this introduces more significant rounding errors. Therefore, it is important to select the quantization parameters to find the right balance between the clipping and rounding errors.

The architecture of Quant-YOLOv5, illustrated in Fig. 4, represents an optimized and quantized version of the YOLOv5 model, designed to reduce latency and enhance computational efficiency during inference while maintaining detection accuracy. Several modifications were made to achieve these objectives.

1) Simplifying activation functions: The Sigmoid Linear Unit (SiLU) function is used as the activation function in YOLOv5. This activation function, although useful in specific neural network applications, has significant issues when employed with quantized YOLOv5 models, especially for deployment on edge devices. SiLU's lack of quantization friendliness and inadequate support in embedded systems is a notable constraint. The incompatibility stems from the computational complexity and continuous characteristics of SiLU, which complicate the quantization process and may result in an

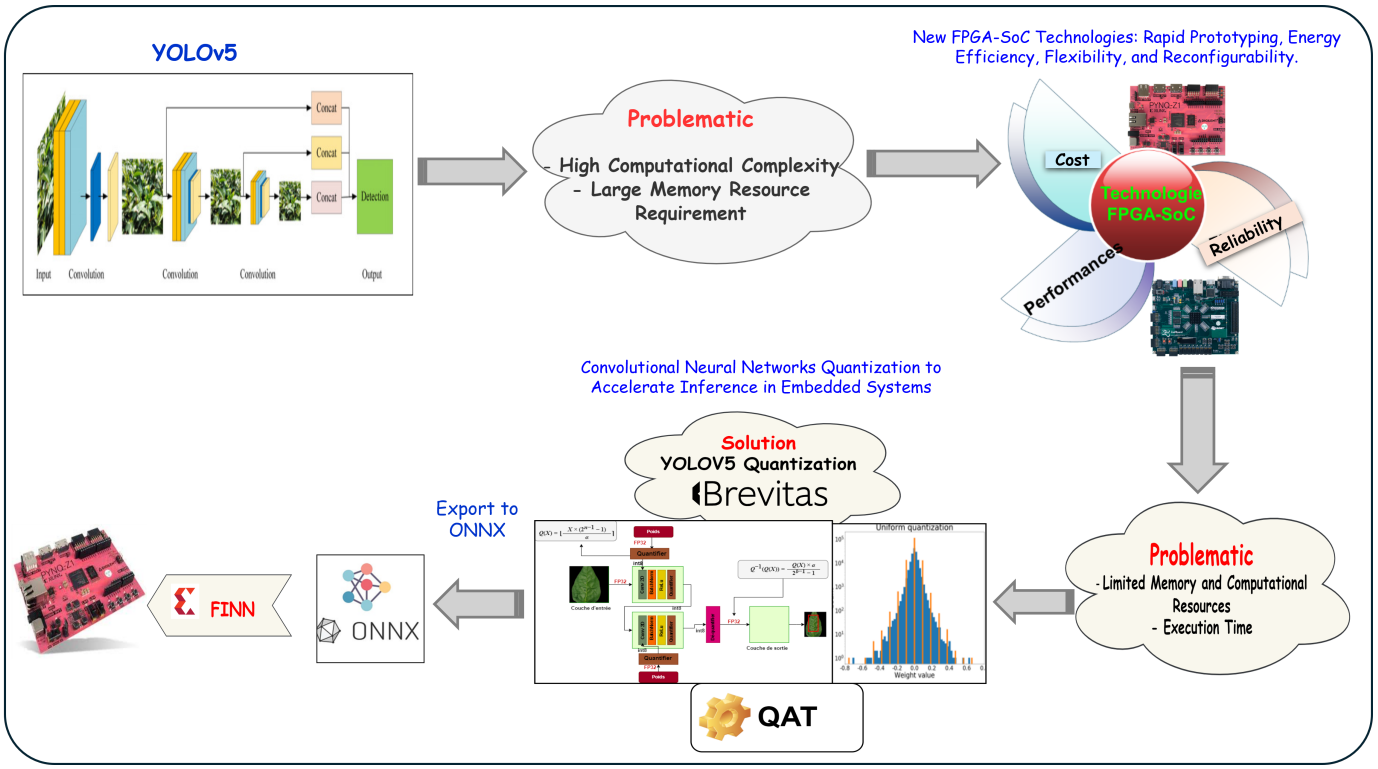

Fig. 3. Proposed methodology overview.

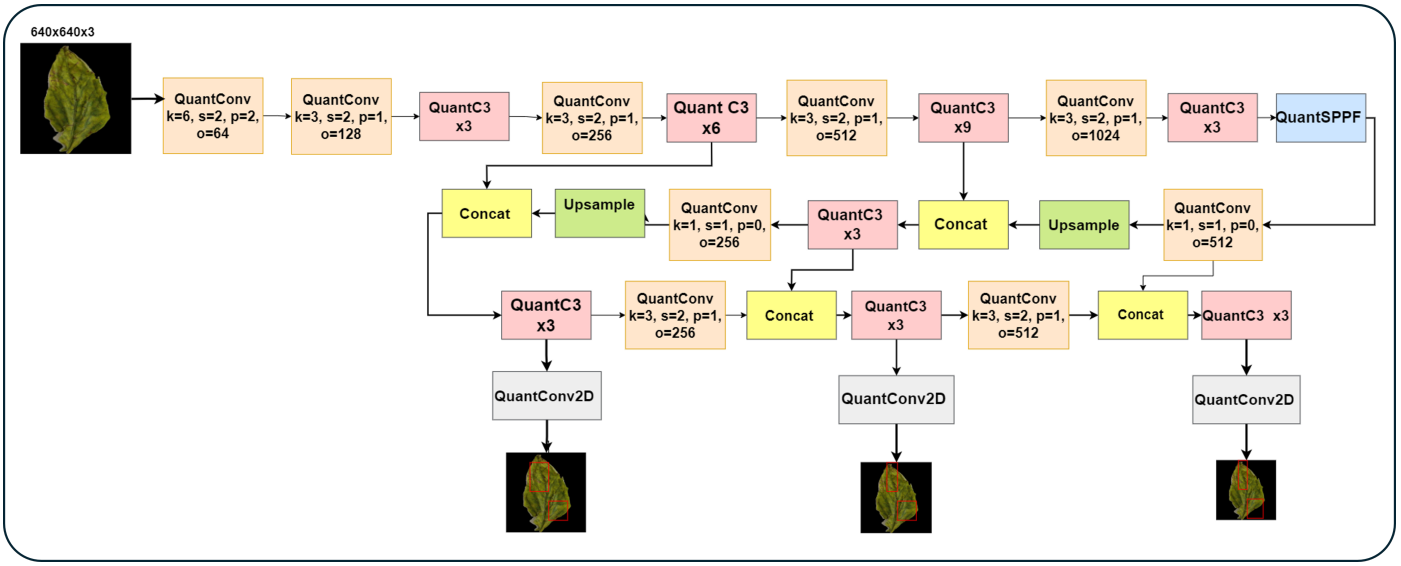

Fig. 4. Architecture of quantized YOLOv5.

overall decrease in accuracy. Furthermore, when implementing logarithmic scale quantization for activation functions that yield negative values, like SiLU, it is essential to consider the absolute value of the input. This modification leads to a distribution that diverges considerably from the initial one, which may result in a notable reduction in the accuracy of the model. To overcome this limitation, we suggest replacing SiLU with Rectified Linear Unit (ReLU). This substitution greatly minimizes computational complexity without compromising activation performance. ReLU, being a simpler and faster function to compute, demonstrates inherent compatibility with quantization and is particularly well-suited for resource-constrained edge devices.

2) Fusing BN and convolution layers: The BatchNormalization (BN) layers have fused with adjacent convolutional layers. We perform this merging to reduce data transfer and unnecessary operations, thus simplifying the model and enhancing its efficiency during inference.

3) Quantized layers: QuantConv and QuantReLU are the quantized versions of the convolutional layer and activation function, respectively, as depicted in Fig. 5. These layers use 8-bit integer (Int8) weights and activations instead of floating point (FP32) values.

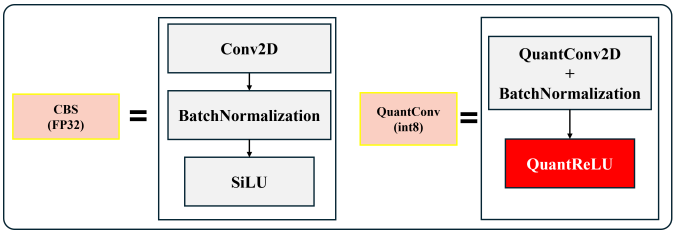

Fig. 5. Comparison of original CONV layer and QuantConv layer.

The C3 and SPPF blocks have been replaced with their quantized equivalents, QuantC3 and QuantSPPF, as shown in Fig. 6 and Fig. 7. The QuantC3 block retains the modular structure of the C3 block but incorporates quantized convolutions and activations. The QuantSPPF module is designed to capture multiscale information through sequential max-pooling operations while maintaining optimal computational efficiency. The max-pooling operations in QuantSPPF are not quantized because the input and output values remain on the same quantization grid, making the quantization of these operations unnecessary.

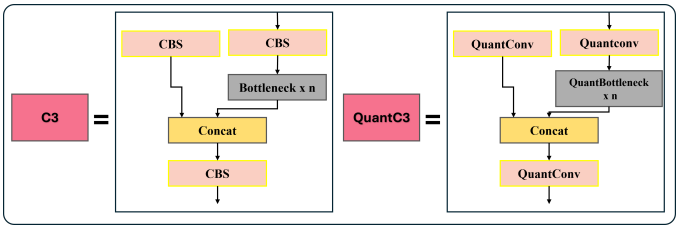

Fig. 6. Comparison of original C3 layer and QuantC3 layer.

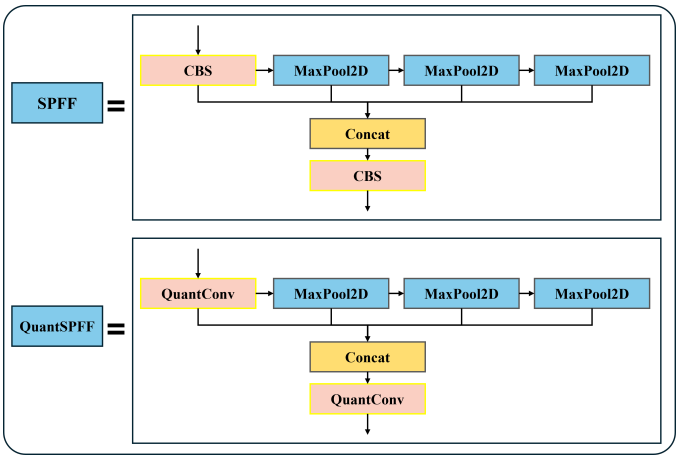

Fig. 7. Comparison of original SPPF layer and QuantSPPF layer.

The upsample and concat operations are not quantized. The upsample, which increases the resolution of feature maps, and Concat, which combines features from different layers, are not quantized to avoid accuracy loss. These operations are implemented using PyTorch functions, which directly manipulate the tensors without introducing additional non-linear computations. Quantifying these operations would not provide significant benefits and could even introduce accuracy errors due to interpolation of values and data manipulation, which would harm the overall quality of the combined features.

In the training phase, the Sigmoid activation function is used in the last layer for better learning. However, for the synthesis phase, this function is replaced by QuantHardtanh, in order to reduce latency and ensure compatibility with the FINN synthesis tool [28].

IV. Experimental Results

A. Dataset

We trained and tested the Quant-YOLOv5x model using a self-generated database. The dataset contains 522 images of both healthy and diseased tomato and pepper leaves [29]. It is divided into six categories: Tomato Yellow Leaf Curl Virus (TYLCV), healthy leaves, leaf miner, mildew, nutrient deficiency, and oidium. All images were captured using a smartphone camera in several tomato and pepper greenhouses located in Monastir, Tunisia, under the guidance of an agricultural engineer. To overcome the challenges caused by shadows and distracting backgrounds—which can reduce detection accuracy by hiding important leaf features—we applied a segmentation process that removes the background and enhances the focus on the relevant leaf regions. Fig. 8 presents examples of the images included in the dataset.

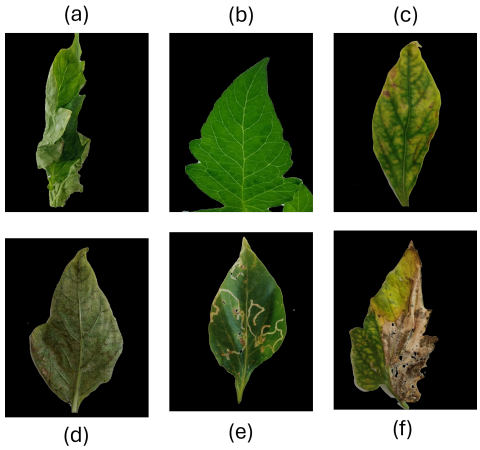

Fig. 8. Examples of dataset images: (a) TYLCV, (b) Healthy, (c) Nutrient Deficiency, (d) Oidium, (e) Leaf Miner, and (f) Mildew.

B. Experimental Settings

YOLOv5x, the largest variant in the YOLOv5 family, offers superior accuracy in object detection tasks due to its increased depth and width, allowing for more precise feature extraction. However, its substantial model size and computational demands pose challenges for deployment on resource-constrained edge platforms, leading to increased latency and higher energy consumption. To address these limitations, we applied quantization techniques to YOLOv5x, aiming to reduce

its size and computational requirements, facilitating efficient deployment on FPGAs without significantly compromising detection performance.

To evaluate the effectiveness of quantization of the YOLOv5x model for the detection and localization of plant disease, we compared the performance of the YOLOv5x and Quant-YOLOv5x models. The YOLOv5x model uses 32-bit floating-point weights (Float32), while the Quant-YOLOv5x model uses 8-bit integer (Int8) weights. Quant-YOLOv5x was trained on a Google Colab Pro GPU. Google Colab Pro provides NVIDIA A100 GPUs, optimized for machine learning tasks with 16 GB GPU memory and FP32 and INT8 computing capabilities. We tested both models on the Google Colab Pro CPU, typically an Intel Xeon processor with a base frequency of 2.3 GHz and up to 4 cores, which offers a balance between computing power and energy efficiency.

C. Results and Discussion

The model was quantized using the Brevitas library. However, it does not perform any low-precision acceleration by itself. To obtain results, the model must first be exported to an inference toolchain via an intermediate representation such as ONNX. As a result, the inference of models in the ONNX format runs mainly on the CPU, which justifies the use of the Google Colab Pro CPU for testing. Table I shows a performance comparison of YOLOv5x and Quant-YOLOv5x using four main metrics: precision, recall, F1-score, and mAP (mean average precision).

TABLE I. COMPARISON OF PERFORMANCE METRICS FOR YOLOV5X AND QUANT-YOLOV5X

Model	Precision (%)	Recall (%)	F1-Score (%)	mAP@50 (%)
Standard YOLOv5x	88.7	92.8	90.7	95.5
Quant-YOLOv5x	73.9	88.5	80.5	93.7

The precision of the YOLOv5x model is 88.7%, significantly higher than that of Quant-YOLOv5x at 73.9%. The quantization of weights causes a loss of precision by reducing the granularity of the represented values, which is responsible for this difference. Furthermore, YOLOv5x has a higher recall rate (92.8% vs. 88.5% for Quant-YOLOv5x), because quantization to Int8 can cause loss of information, making it harder for the model to find all cases of disease. The F1-score, which is the harmonic mean of precision and recall, shows the superiority of YOLOv5x, with 90.7% compared to 80.5% for Quant-YOLOv5x. Quantization negatively affects both precision and recall, resulting in a lower F1-score for Quant-YOLOv5x. Although the mAP of Quant-YOLOv5x is relatively close to that of YOLOv5x (93.7% vs 95.5%), the difference suggests that the quantized model detects objects correctly, although with slightly reduced accuracy. This shows that despite quantization, Quant-YOLOv5x maintains respectable performance, which is crucial for edge applications where computational resources are limited. Fig. 9 illustrates the mAP, precision, and recall metrics plots for the original YOLOv5x model and our quantized Quant-YOLOv5x model.

The latency results for a single frame are shown in Fig. 10. The Quant-YOLOv5x model performs approximately 3 times faster than the original model. As shown in the figure, the histogram shows a comparison of the latency between

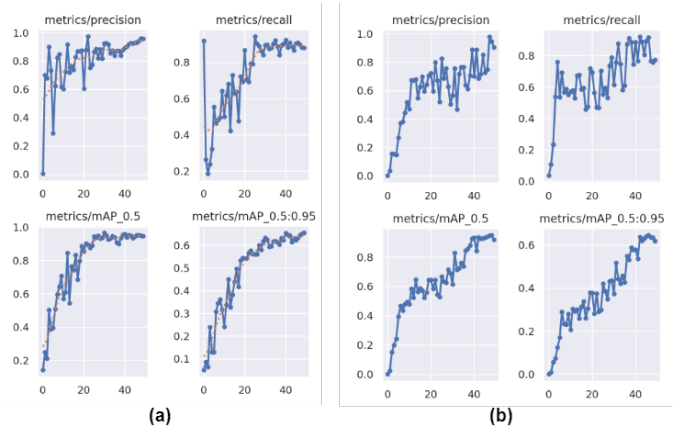

Fig. 9. Precision, recall, and mean average precision (mAP) plots: (a) YOLOv5x model, (b) Quant-YOLOv5x.

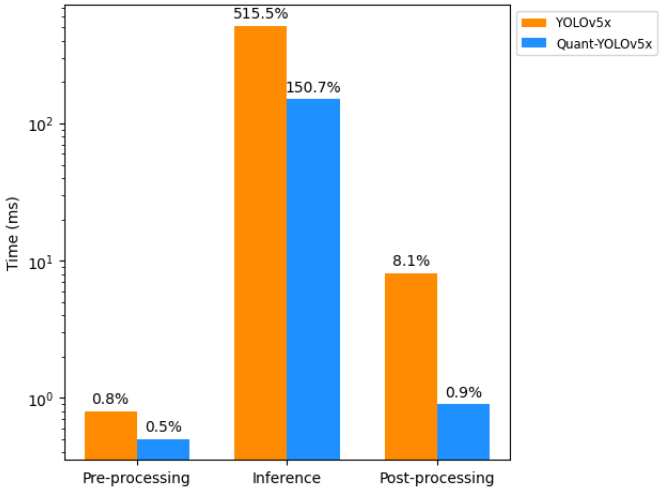

Fig. 10. Comparison of latency.

the two models. The latency is measured in three distinct phases: pre-processing, inference, and post-processing. The total processing time for YOLOv5x, which includes pre-processing, inference, and postprocessing, is 524.4 ms. In comparison, the total processing time for Quant-YOLOv5x is 152.1 ms. This significant reduction in overall latency demonstrates the effectiveness of quantization in terms of processing speed. More specifically, quantization reduces the computational times required for each processing step.

In terms of frame rate per second (FPS), YOLOv5x achieves approximately 1.9 FPS, while Quant-YOLOv5x achieves approximately 6.6 FPS. This shows that the quantized model is capable of processing more frames per second, making Quant-YOLOv5x more suitable for applications that require rapid response times. Quantifying the YOLOv5x model in Quant-YOLOv5x reduced the total processing latency, which is essential for edge computing applications where processing speed is necessary.

Fig. 11 presents a size comparison between the YOLOv5x and Quant-YOLOv5x models. YOLOv5x has a size of 170 MB, while Quant-YOLOv5x has a size of 83 MB. This

TABLE II. COMPARISON OF QUANTIZATION METHODS FOR YOLOV5

Ref	Quantization	Model	Dataset	Precision	Δ mAP50	Δ Size (MB)	Δ FPS
[30]	PTQ	YOLOv5s	COCO	W8A8	-0.2%	6.7	25.8
[31]	QAT	YOLOv5s	Custom	W8A8	-1.9%	26.61	145
[32]	QAT	YOLOv5x	COCO2017	W8A8B8	-0.61%	-	4
Our model	QAT	YOLOv5x	Custom	W8A8	-2.8%	87	4.7

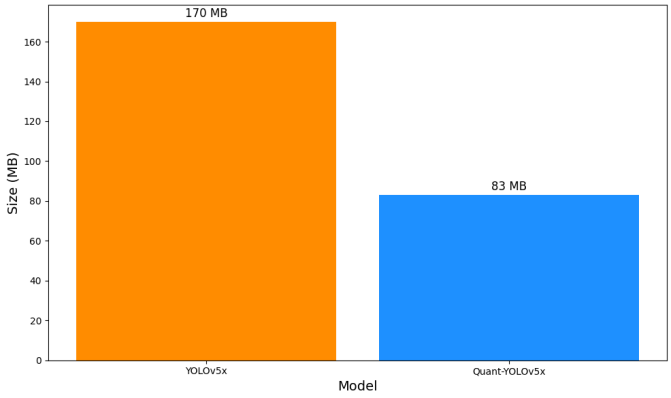

Fig. 11. Comparison of models size.

reduction of more than 50% in model size demonstrates the effectiveness of quantization in terms of reducing the required memory. The reduction in model size is particularly important for deployments on edge devices, where memory and storage resources are limited. By quantizing the weights to Int8, the Quant-YOLOv5x model becomes not only faster but also more compact, facilitating its deployment on resource-constrained platforms.

D. Comparative Study

Table II illustrates a comparison between our proposed quantized Quant-YOLOv5x model and various quantization methods from the literature for YOLOv5 models. The comparison is based on the differences (Δ) recorded in mAP50, the size of the model and the FPS, where Δ represents the variation between the values before and after quantization. The authors of [30] propose a low bit quantization method (W8A8) for YOLOv5, using post-training quantization (PTQ). The Quant-YOLO method uses one-sided histogram (UH)-based quantization to minimize the quantization error of activations. Experiments on the COCO dataset show that Quant-YOLO maintains an average accuracy (mAP50) comparable to that of the full-precision model, while reducing the model size from 14.4 to 7.7 MB and increasing the FPS from 25.8 to 42.7 frames per second. In [31], a QAT method is proposed with TensorFlow Lite to quantify the weights and activations of the 8-bit precision YOLOv5 model for fire detection, using a custom database. The quantized YOLOv5 model achieves a mAP50 of 95.9%, with a performance loss of 1.9% compared to the 32-bit floating point (FP32) YOLOv5 model. Furthermore, the size of the 8-bit YOLOv5 model is reduced to 13.1 MB, which is a decrease of 26.61 MB compared to the FP32 version. Finally, the FPS increases significantly, from 2 to 147 frames per second. The study in [32] presents a method named USPIQ, which combines the advantages of in-training and out-training quantization to quantize the YOLOv5 model. USPIQ quantizes all internal floating-point operations (weights, activations, biases) into integer operations. The YOLOv5x model was evaluated using the COCO val2017 database. The floating-point (FP32) model achieves an mAP50 of 67. 31%, while the 8-bit model maintains an mAP of 66. 7%. In addition, the FPS of the 8-bit model is improved, from 2.17 FPS for the FP32 model to 6.17 FPS.

Our model, Quant-YOLOv5x, proposes a QAT approach for YOLOv5x using a custom dataset. We achieved an accuracy loss of 2.8%, a model size reduction of 87 MB, and an increase in frames per second of 4.7. Although the accuracy loss is slightly higher than that of other methods, our approach stands out with a significant reduction in model size, making our solution particularly suitable for resource-constrained devices and real-time applications.

V. CONCLUSION

In this paper, we addressed the challenges associated with the deployment of convolutional neural networks (CNN) on FPGAs. To overcome the limitations of YOLOv5x implementation, we proposed a quantization technique optimized using the Brevitas library, enabling a more efficient and resource-conscious execution. The quantization results demonstrate that Quant-YOLOv5x significantly reduces both latency and model size, making it highly suitable for edge computing applications where computational and memory resources are constrained. Despite a slight degradation in detection performance, Quant-YOLOv5x remains effective and well-suited for edge deployments. As future work, we aim to focus on the practical implementation of our method to validate its effectiveness and performance under real-world conditions. Additionally, we investigate the integration of Quant-YOLOv5x with other edge AI frameworks to improve its adaptability across a wider range of hardware platforms.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Research and Graduate Studies at King Khalid University for funding this work through a Large Group Project under grant number (RGP2/473/46).

REFERENCES

[1] Ministry of Agriculture, Water Resources, and Fisheries of Tunisia: Key Indicators of Tunisian Agriculture in 2023, Available: http: www.onagri.nat.tn/uploads/IndicateursclesdelagricultureVF(2023).pdf (2023)

[2] Interprofessional Group of Vegetables of Tunisia, *Pepper*, Available: https://www.gil.com.tn/fr/product?label=piment_16, 2015.

[3] Tunisie Numerique, *Agriculture: Challenges and Issues in the Tomato Processing Sector in Tunisia.* Available: https:www.tunisienumerique.com, 2024.

[4] N. Buzkan, B. B. Arpaci, V. Simon, H. Fakhfakh, and B. Moury, "High prevalence of poleroviruses in field-grown pepper in Turkey and Tunisia," *Archives of Virology*, vol. 158, pp. 881–885, 2013. [Online]. Available: Springer.

[5] Jocher, G., Stoken, A., Chaurasia, A., Borovec, J., Kwon, Y., Michael, K., Changyu, L., Fang, J., Skalski, P., Hogan, A., et al.: ultralytics/yolov5: v6. 0-yolov5n'nano'models, roboflow integration, tensorflow export, opencv dnn support. Zenodo (2021)

[6] Wang, C.-Y., Liao, H.-Y.M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., Yeh, I.-H.: Cspnet: A new backbone that can enhance learning capability of cnn. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 390–391 (2020)

[7] Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path aggregation network for instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8759–8768 (2018)

[8] Mi, J.-X., Feng, J., Huang, K.-Y.: Designing efficient convolutional neural network structure: A survey. Neurocomputing 489, 139–156 (2022)

[9] Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing 461, 370–403 (2021)

[10] Li, R., Wang, Y., Liang, F., Qin, H., Yan, J., Fan, R.: Fully quantized network for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2810–2819 (2019)

[11] Nagel, M., Baalen, M.v., Blankevoort, T., Welling, M.: Data-free quantiza- tion through weight equalization and bias correction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1325–1334 (2019)

[12] Zeng, T., Li, S., Song, Q., Zhong, F., Wei, X.: Lightweight tomato real-time detection method based on improved yolo and mobile deployment. Computers and electronics in agriculture 205, 107625 (2023)

[13] PZhang, J., Zhou, Y., Saab, R.: Post-training quantization for neural networks with provable guarantees. SIAM Journal on Mathematics of Data Science 5(2), 373–399 (2023)

[14] Banner, R., Nahshan, Y., Soudry, D.: Post training 4-bit quantization of convolu- tional networks for rapid-deployment. Advances in Neural Information Processing Systems 32 (2019)

[15] Rifqie, D.M., Surianto, D.F., Abdal, N.M., Hidayat, W., Ramli, H.: Post training quantization in lenet-5 algorithm for efficient inference (2022)

[16] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision, pp. 291–326. Chapman and Hall/CRC,(2022)

[17] Wu, Q., Li, Y., Chen, S., Kang, Y.: Drgs: Low-precision full quantization of deep neural network with dynamic rounding and gradient scaling for object detection. In: International Conference on Data Mining and Big Data, pp. 137–151 (2022). Springer

[18] Papaioannou, A., Kouzinopoulos, C.S., Ioannidis, D., Tzovaras, D.: An ultra-low- power embedded ai fire detection and crowd counting system for indoor areas. ACM Transactions on Embedded Computing Systems 22(4), 1–20 (2023).

[19] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016).

[20] Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.-J., Srinivasan, V., Gopalakr- ishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085 (2018)

[21] MEsser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S.: Learned step size quantization. arXiv preprint arXiv:1902.08153 (2019)

[22] Blott, M., Preußer, T.B., Fraser, N.J., Gambardella, G., O'brien, K., Umuroglu, Y., Leeser, M., Vissers, K.: Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 11(3), 1–23 (2018)

[23] Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: Finn: A framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-programmable Gate Arrays, pp. 65–74 (2017)

[24] Goez, D., Soto, P., Latre, S., Gaviria, N., Camelo, M.: A methodology to design quantized deep neural networks for automatic modulation recognition. Algorithms 15(12), 441 (2022).

[25] Reiter, P., Karagiannakis, P., Ireland, M., Greenland, S., Crockett, L.: Fpga acceleration of a quantized neural network for remote-sensed cloud detection. In: 7th International Workshop on On-Board Payload Data Compression (2020).

[26] A. Pappalardo, "Xilinx/brevitas," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.3333552.

[27] Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)

[28] G¨unay, B., Okcu, S.B., Bilge, H.S¸.: Lpyolo: low precision yolo for face detection on fpga. arXiv preprint arXiv:2207.10482 (2022)

[29] Tej, B., Bouaafia, S., Hajjaji, M.A., Mtibaa, A.: An Improved YOLOv5 for Accu- rate Detection and Localization of Tomato and Pepper Leaf Diseases. Available: https://doi.org/10.21203/rs.3.rs-3358463/v1

[30] Wang, M., Sun, H., Shi, J., Liu, X., Cao, X., Zhang, L., Zhang, B.: Q-yolo: Efficient inference for real-time object detection. In: Asian Conference on Pattern Recognition, pp. 307–321 (2023). Springer.

[31] Papaioannou, A., Kouzinopoulos, C.S., Ioannidis, D., Tzovaras, D.: An ultra-low- power embedded ai fire detection and crowd counting system for indoor areas. ACM Transactions on Embedded Computing Systems 22(4), 1–20 (2023)

[32] Al-Hamid, A.A., Kim, H.: Unified scaling-based pure-integer quantization for low- power accelerator of complex cnns. Electronics 12(12), 2660 (2023)