Quantization Strategies for Low-Resource Hardware in Deep Learning Applications

Caio Martins de Abreu

Abstract

This work investigates quantization techniques applied to computational systems with limited hardware resources, focusing on machine learning and signal processing applications. Quantization allows for the representation of numerical data with reduced precision, significantly decreasing memory usage and computational cost. Such techniques are essential for embedded and energy-constrained devices. This study explores and evaluates different quantization strategies to balance accuracy, efficiency, and hardware compatibility.

1 Introduction

The exponential growth of deep neural network (DNN) applications, especially in computer vision, has increased the demand for deploying such models on embedded platforms like Field-Programmable Gate Arrays (FPGAs). Although FPGAs offer high performance and flexibility, implementing full-precision models (e.g., 32-bit floating-point) on these platforms is often inefficient in terms of power consumption, logic utilization, and memory bandwidth.

In this context, post-training quantization (PTQ) has emerged as a promising solution to reduce model complexity without requiring retraining. Among the PTQ techniques, methods like Accurate and Hardware-Compatible Post-Training Quantization (AHCPTQ) have shown the ability to preserve model accuracy while enabling compatibility with hardware-optimized arithmetic, which is critical for FPGA-based implementations.

Additionally, trainable fixed-point quantization approaches, such as those proposed by Lin et al., demonstrate that integrating quantization into the training process can enhance computational efficiency without degrading predictive performance. This strategy leverages FPGA flexibility in defining data paths and word lengths to create hardware-aware models.

Furthermore, replacing standard floating-point formats with reduced-precision representations — such as half-precision (16-bit) or custom floating-point types — has also been explored to improve throughput and energy efficiency in FPGA-based accelerators.

This work proposes the evaluation and implementation of quantization techniques on convolutional neural networks, focusing on strategies that balance accuracy, computational efficiency, and hardware compatibility. The goal is to enable efficient inference in real-world embedded AI scenarios.

2 Methodology

This study adopts an experimental approach to evaluate the impact of post-training quantization techniques on convolutional neural networks (CNNs) targeting embedded hardware. Two widely used vision models were selected for analysis: the *Segment Anything Model* (SAM) by Meta AI, and *You Only Look Once* (YOLO). The focus was on preserving predictive quality after quantization for deployment in resource-constrained environments.

2.1 Selected Models

SAM was selected for its versatility in image segmentation, while YOLO represents a well-established solution for real-time object detection. These models serve as representative benchmarks for assessing the behavior of quantization across distinct vision tasks.

2.2 Quantization Techniques

Post-training quantization (PTQ) was applied using native tools from the PyTorch framework, without altering model architectures.

1. *Dynamic quantization (SAM):* applied to weights using 8-bit integers (INT8), while activations remain in floating point. Quantization occurs at inference time and requires no calibration.

2. *Static quantization (YOLO):* applied to both weights and activations, using a calibration dataset to determine optimal quantization ranges. This approach offers greater efficiency gains but may impact accuracy more significantly.

2.3 Experimental Procedure

Quantization was performed directly on pre-trained models using PyTorch 2.6. All experiments were conducted in a controlled software environment, emulating conditions similar to those found in embedded hardware platforms.

2.4 Evaluation Criteria

A qualitative evaluation was conducted based on the following criteria:

1. Visual quality of the outputs (segmentation masks from SAM and detection boxes from YOLO);

2. Semantic coherence between input and prediction;

3. Structural consistency compared to the original full-precision models.

Although no formal quantitative metrics (e.g., IoU or mAP) were applied, the qualitative assessment provided sufficient insight into the practical impact of quantization on perceptual performance and deployment feasibility.

2.5 Tools

All experiments were carried out using PyTorch 2.6, leveraging its built-in quantization features. The models were kept intact to ensure compatibility with export formats such as ONNX, facilitating future integration with FPGA synthesis flows.

3 Quantization Workflow

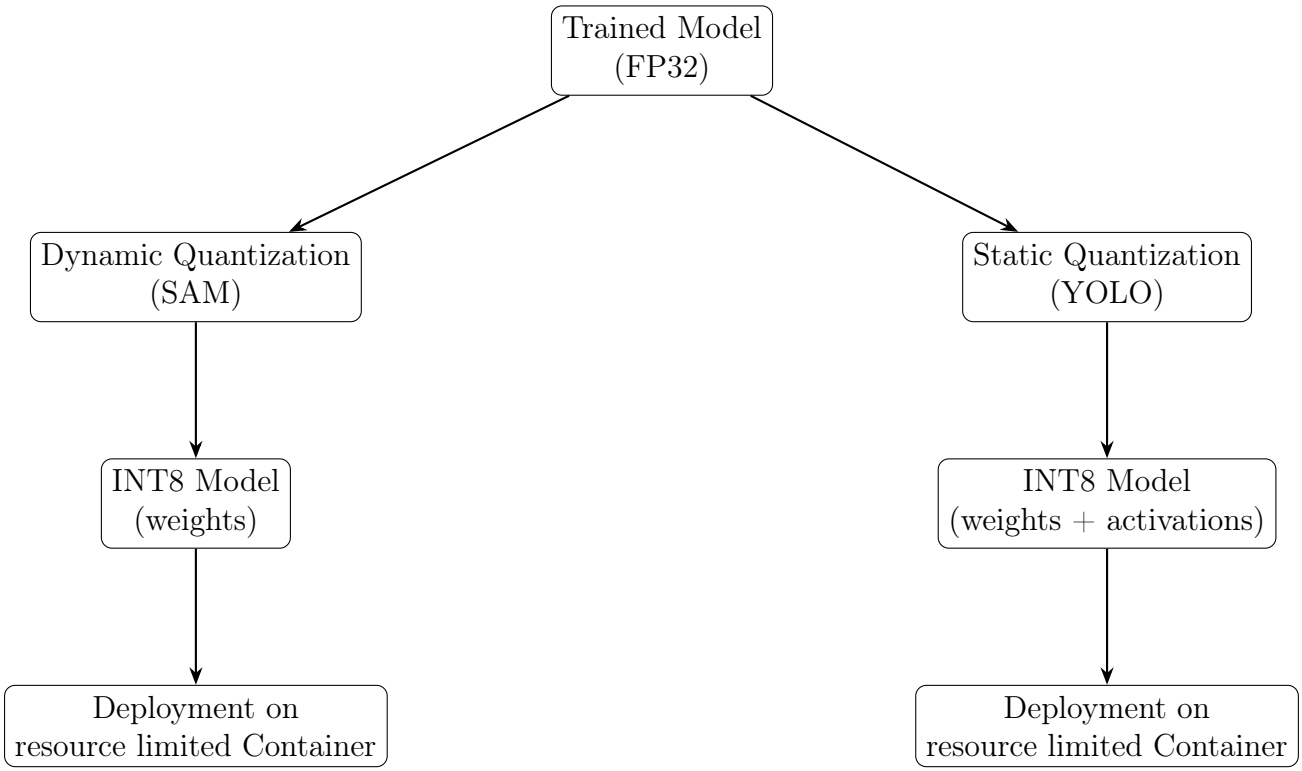

Figure 1: Quantization workflow for SAM and YOLO using PyTorch

4 Results Overview

Table 1: Qualitative comparison before and after quantization

Model	**Precision**	**Inference Speed**	**Visual Quality**
SAM (Original)	FP32	Low	High
SAM (Quantized)	INT8 (weights)	Moderate	High
YOLO (Original)	FP32	Moderate	High
YOLO (Quantized)	INT8 (weights + activations)	High	Slightly reduced