Bringing CNN Inference at the edge through quantization techniques

Nicolò Vacis
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano
Milan, Italy
10765405

***Abstract*—Convolutional Neural Networks (CNNs) are fundamental in modern computer vision and deep learning applications, but their high computational and memory requirements pose significant challenges for deployment on resource-constrained embedded Field-Programmable Gate Arrays (FPGAs). This project explores quantization techniques to optimize CNN inference on embedded FPGA platforms, aiming to enhance computational efficiency while maintaining model accuracy. Quantization methods are analyzed to lower precision requirements, enabling faster execution with reduced power consumption. The objective of the study is to evaluate the trade-offs between accuracy, performance, and hardware utilization on an embedded FPGA board. Relevant code is available at https://github.com/nicolovacis/cnn-edge-quantization**

I. Introduction

Edge applications often require real-time decision-making under tight latency and energy constraints. Our goal was to assess whether a surface might be damaged, based on the light reflection captured by an image photosensor. The optical principle behind this process is detailed in Section II-A. Accurate and hardware efficient estimation of this reflection is essential when operating in environments with limited computational resources.

Previous approach and its limitations. Previously, the system relied on a traditional optical sensor pipeline combined with a classical algorithm. It estimated the diameter of the circular reflection to assess the sensor-to-surface alignment. While this method proved sufficient performance in ideal conditions, it exhibited significant limitations in edge cases—particularly when the reflection was partially visible, distorted, or occluded. Under these conditions, its performance degraded considerably, leading to inaccurate assessments and poor reliability.

Our solution. To solve the previous limitations, we designed a new sensor pipeline that incorporates multiple deep learning models optimized for edge inference on embedded FPGA platforms. The solution introduces a two-stage decision system: a lightweight classification model first determines whether the input conditions are favorable for traditional ML-based estimation. If not, a second, more robust CNN-based regression model is used to directly estimate the diameter of the reflection.

Both models are quantized using Brevitas and synthesized through the FINN toolchain to ensure compatibility with the target FPGA hardware. This enables efficient, low-latency execution suitable for edge applications in constrained environments.

Objective and evaluation. The main goal of this work is to explore how quantized deep learning models can be deployed on embedded FPGAs to enhance robustness and accuracy in surface analysis tasks. The evaluation focuses on a hardware-aware hyperparameter tuning process that jointly considers model accuracy (for classification), average error (for regression), hardware utilization (LUTs, BRAMs, FF), and throughput. By analyzing these trade-offs, we aim to identify the optimal configurations that maintain predictive performance while meeting edge deployment constraints.

II. Background

The purpose of this section is to walk through the fundamentals of the new sensor pipeline. We begin with the optical probe and then move through the acquisition chain and signal processing steps that enable accurate interpretation of the data.

A. Optical Probe

The sensor operates based on an optical reflection principle used to assess whether a target surface is properly aligned and suitable for analysis. The system consists of an LED light source that emits a beam through a circular slit, which is then projected onto the target surface via an optical assembly. The reflection of this beam forms a circular pattern that is captured by an image sensor.

The key parameter extracted from this setup is the diameter of the reflected ring. This diameter depends on the distance and orientation of the surface with respect to the sensor. When the surface is within the expected range and aligned correctly, the ring appears well-defined and symmetrical on the sensor. Deviations in size or shape indicate variations in distance, surface irregularities, or misalignment.

This physical principle forms the foundation of the surface validation process, allowing the system to make geometric assessments based on visual light behavior, independently of surface material or texture.

B. Deep Learning - CNN

Deep learning is a branch of machine learning that uses multi-layer neural networks to learn complex patterns from

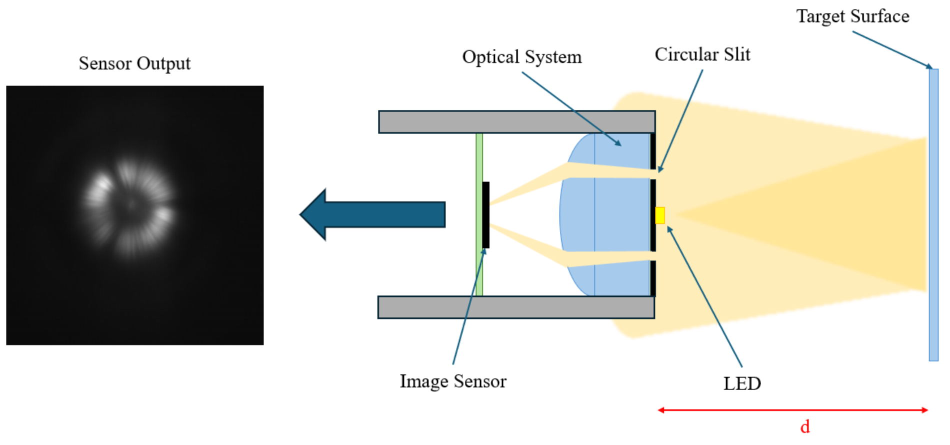

Fig. 1. Sensor setup: LED light passes through a circular slit and optical system, projecting a ring-shaped reflection onto the image sensor. The diameter of this ring provides information on surface alignment and suitability.

data. In particular, Convolutional Neural Networks (CNNs) are well-suited for computer vision tasks, as they perform well at interpreting spatial patterns from images through layers of convolution, pooling, and activation [1].

In this project, CNNs are used to analyze the circular light reflection captured by the sensor, either by classifying the suitability of the image or by regressing the diameter of the ring. This allows the system to perform geometric analysis directly from visual input, enabling fast and reliable decision-making when deployed on FPGAs through quantized, hardware-aware models.

C. CNN on Hardware: Frameworks and Optimization

Recent advances in hardware platforms support the deployment of Convolutional Neural Networks, including GPUs, TPUs, NPUs, and FPGAs. GPUs are effective for high-throughput inference and training due to their general-purpose parallelism, while TPUs and NPUs are optimized for neural computations, offering efficient matrix operations and low-latency performance.

FPGAs are well suited for edge scenarios thanks to their reconfigurability and fine-grained control over computation and memory. This enables highly optimized dataflows with benefits in energy efficiency, compact area usage, and lower cost. However, FPGAs are constrained by limited on-chip resources and typically offer lower peak throughput compared to GPUs.

The choice of hardware depends on the deployment context: FPGAs are better suited for embedded and real-time systems—such as IoT devices and portable sensors—where size, power, and costs are critical. GPUs remain preferable in data center environments for large-scale batch processing and training workloads, where power and space are less restrictive.

Quantization-aware toolchains is then introduced to efficiently map CNN models to FPGA architectures. The first key component in our pipeline is the use of the Brevitas library. Brevitas is a PyTorch-based framework designed for quantization-aware training (QAT), which allows networks to be trained with customized bit-widths for weights, activations, biases, and Input/Output quantizations. QAT simulates the effects of quantization during training, enabling the network to adapt to the reduced numerical precision that will encounter during deployment. In Brevitas, this is implemented using simulated quantization: tensors are quantized to the desired bit-width (e.g., 8-bit), but all computations are internally performed in floating-point (float32) during both forward and backward passes. Once trained, the model is exported to the ONNX format, where all quantization parameters and structure are explicitly encoded [2]. This ONNX graph serves as the entry point for hardware synthesis using the FINN toolchain.

Secondly, we introduced FINN (open-source framework developed by AMD/Xilinx) to convert these models into synthesizable hardware dataflows, allowing the tuning parallelism, and resource allocation from quantized neural networks. It consumes the quantized ONNX model exported by Brevitas and transforms it into a pipeline of hardware accelerators specifically tailored for FPGA deployment. At this stage, simulated quantization from the training phase is replaced by actual low-precision arithmetic: all operations are implemented using true integer computation. This marks the transition from quantization-aware simulation to real hardware-level quantization. FINN offers fine-grained control over a range of architectural parameters that directly impact performance and resource usage. These include parallelization settings such as the number of PE and SIMD known as folding factors—which define how computations are distributed across hardware to maximize throughput. PEs determine how many output channels are computed in parallel, while SIMD controls the number of input channels processed per cycle within each PE. Additionally, FIFO depths regulate the buffering of data between layers to prevent pipeline stalls. By tuning these parameters, designers can efficiently optimize models and extract maximum performance under the given resource constraints (LUTs, BRAMs, and FFs). This process is essential for balancing performance with the constraints of embedded

FPGA deployment [3].

An alternative to the FINN tool is Vitis-AI, a development stack by AMD/Xilinx designed to deploy neural networks on pre-architected Deep Processing Units (DPUs). It supports a wide range of models and automates processes like quantization and compilation, simplifying deployment.

Compared to FINN, Vitis-AI offers less flexibility in hardware customization. While FINN allows for detailed control over architectural parameters—such as processing element allocation, SIMD configuration, and memory mapping—Vitis-AI abstracts many low-level details to prioritize ease of use and portability. This makes FINN more suitable for applications requiring fine-tuned hardware optimization, especially under strict resource constraints [4].

The ability to tune parameters in both Brevitas and the hardware synthesis toolchains (Hardware Hyperparameter Tuning) is essential for navigating the trade-off between prediction performance and hardware resource utilization. By systematically adjusting these settings, we can tailor each model variant to fit within the constraints of a target FPGA while maximizing throughput and maintaining acceptable accuracy.

III. Methodology

The methodology follows a structured pipeline, beginning with data preparation and quantization-aware training, and progressing through model export, hardware synthesis, and deployment on the target FPGA.

A. General Idea

The proposed pipeline follows a quantization-aware workflow combining Brevitas for training and FINN for hardware synthesis targeting the Ultra96v2 platform. Starting from standard CNN architectures in PyTorch or TensorFlow, we introduce quantized layers using Brevitas and train models for both classification and regression. After training, quantized ONNX models are exported and passed through the FINN toolchain to generate synthesizable dataflow architectures.

The main limitation encountered during development was related to **FIFO buffer allocation**. The FINN toolchain, when targeting the Ultra96v2 platform, consistently overestimated the depth of the FIFO buffers between layers, with no actual check against the hardware's capability to support such long buffers. As a result, synthesis failed due to infeasible resources allocation.

We also attempted to manually set a maximum FIFO depth during the build to confirm the previous behavior. Although this bypassed the auto-allocation issue, the resulting design failed during post-synthesis throughput tests on the board, showing that the underlying resource constraints remained unsatisfied.

To verify whether the issue was strictly due to board limitations, we replicated the same experiments on a larger FPGA (U55C). In this case, FINN completed synthesis without errors. However, we note that the Ultra96v2 platform has limited resources and is not a primary target architecture for FINN, which appears to be less structured or optimized for such constrained devices compared to larger, more commonly supported platforms.

As a consequence, several architectural simplifications were necessary to enable deployment on the Ultra96v2:

- **Input size reduction:** Models were progressively downscaled from 300×300 to 128×128 to reduce feature map sizes throughout the network.
- **Conv and dense simplification:** The number of filters in convolutional layers and neurons in dense layers was significantly reduced to avoid exceeding the resource limits and ensure feasible synthesis.
- **PE and SIMD constraints:** FINN's folding factors (parallelism parameters) had to be fixed at one—i.e., minimum parallelization—to avoid FIFO-related synthesis failures.

All these factors—input size, convolutional depth, dense layer width, and folding factors (PE/SIMD)—had to be scaled down to fit within the synthesis budget of the Ultra96v2. As a result, compact architectures were adopted, leading to significantly lower hardware cost, but at the expense of reduced predictive performance—especially for the regression task. These trade-offs are analyzed in detail in the following Results section.

1) *Classification*

The binary classification model was designed to decide whether the acquired image was suitable for traditional estimation or required further analysis through a CNN-based regression stage.

The original classification model (designed in TensorFlow, PyTorch and Brevitas) took as input 300×300 images, with multiple convolutional layers followed by dense layers.

The model was re-implemented in Brevitas and redesigned to support input sizes of 128×128 and 64×64 to enable FPGA deployment. The network was also structurally simplified: the number of convolutional layers and channels was reduced, and the fully connected stages were significantly minimized. This downsizing was crucial to avoid exceeding the neuron threshold that could trigger FIFO allocation failures during FINN synthesis.

It also helped the classification network successfully complete the FINN hardware generation process and enabled deployment on the target FPGA board with similar performance.

2) *Regression*

The regression model was responsible for directly estimating the diameter of the reflected light ring, which is crucial for assessing surface alignment when input conditions are suboptimal.

Originally, it followed a deeper and wider structure with input size 300×300.

However, the synthesis constraints imposed by FINN—particularly related to FIFO buffer allocation and overall resource usage—necessitated a substantial simplification of the original network architecture. The revised regression model eliminates one convolutional layer, significantly reduces the number of channels across all stages, and employs over 10 times fewer neurons in the fully connected layers. Additionally, it operates on downsized inputs of 128×128 or 64×64 and avoids wide intermediate tensors, ensuring compatibility with the FPGA's limited LUT and BRAM capacity.

This reduction enables efficient synthesis and execution on hardware, but introduces coarser output resolution and lower regression accuracy.

B. HW-aware Hyperparameter Tuning

A hardware-aware Hyperparameter Tuning tuning was integrated in the pipeline in order to further optimize the deployment. This process systematically explored multiple architectural configurations and quantization schemes to identify optimal trade-offs between model performance and hardware feasibility.

The following parameters were varied during experimentation:

- **Bit-widths for weights and activations:** tested values include 2, 4, 6, and 8 bits.
- **Input/output quantization:** we introduced a binary switch to optionally apply quantization at the input and output levels using `Int8ActPerTensorFloat`.
- **Input size:** 64×64 and 128×128 were used; larger sizes (e.g., 300×300) were tested but systematically failed due to FIFO resource overflows.
- **Filter scaling:** convolutional layer widths were controlled via a multiplicative factor, producing structures such as 8–12–16–20–24 or, for more demanding configurations, 16–24–32–40–48 filters across layers.
- **FINN-specific parameters (PE/SIMD/FIFO):** parallelism (PE and SIMD) was fixed to 1 due to synthesis constraints on the Ultra96v2.

Each configuration was synthesized and profiled using FINN's reporting tools to extract hardware metrics such as BRAM usage, LUT and Flip Flop count. These results were used to construct a design space and identify viable architectures that balanced accuracy with resource constraints.

The pipeline concludes with the generation of a deployable bitstream tested directly on the Ultra96v2 platform, validating the throughput under realistic deployment conditions.

IV. Results

In order to guide efficient deployment, the following analysis compares design variants, highlighting trade-offs between predictive accuracy and FPGA resource usage.

A. Experimental Setup

For the classification task, the dataset consisted of 8,814 labeled samples, split into 70% for training, 10% for validation, and 20% for testing. This choice was made to slightly emphasize test reliability across different model configurations. In the regression setup, a reduced dataset of up to 12,100 images was used, adopting an 80–5–15 split to maintain a larger training base while still ensuring model generalization. In the regression scenario, label normalization was applied prior to training in order to mitigate the impact of outliers.

Despite the flexibility offered by quantization-aware training, the design of both classification and regression models was strongly influenced by the resource limitations of the Ultra96-V2 platform. The constrained hardware budget required a careful reduction of input image size and model complexity. The limited on-chip memory of 216 BRAM blocks (approximately 0.95 MB), together with the maximum of 140,000 flip-flops, placed a filter on the depth and parallelism of the implemented architectures. More critically, the limited availability of logic resources (around 70,000 LUTs) imposed an even stricter constraint on the design of the network architecture. As a result, feature map sizes, number of filters, and intermediate buffer depths had to be reduced to fit within the synthesis and deployment limits. These constraints directly shaped the model architecture, requiring a balance between computational expressiveness and hardware feasibility.

B. Trade-Off Analysis

The experiments conducted highlight the trade-offs between model accuracy, throughput, and hardware resource usage when deploying quantized CNNs on embedded FPGA platforms. Both classification and regression models were evaluated across various architectural configurations and quantization levels.

Note: Empty boxes in the plots represent configurations that exceeded available FPGA resources due to large input size or excessive number of filters. Also, Flip-Flops and BRAMs were not displayed as LUTs were our critical resource, which caused the design to exceed hardware limits.

1) Classification Results

Accuracy and F1: The classification task showed strong robustness to quantization. The best configurations achieved F1 scores over 90% while staying within the LUT limits of the Ultra96v2. Accuracy ranged from 96–97%, and F1 from 0.83 to 0.87, showing comparable performance between the original 300×300 TensorFlow model and quantized Brevitas versions at 128×128 or 64×64.*See Figure 2. See Figure 3.*

LUT Usage: Classification LUT usage increases proportionally with input size, filter count, and weight bit-width. Lowering these parameters reduces LUT usage effectively. *See Figure 4.*

Throughput: Throughput is primarily influenced by input resolution and filter count, with minimal impact from quantization. Reducing input size by half while doubling filters maintains approximately the same throughput. *See Figure 5.*

Summary: Classification models can tolerate aggressive quantization and architectural simplification without significant accuracy degradation. Trade-offs manifest mostly in throughput and hardware resource usage.

2) Regression Results

Average Error (μm): Regression accuracy is much more sensitive to quantization. Lower input sizes, filter reductions,

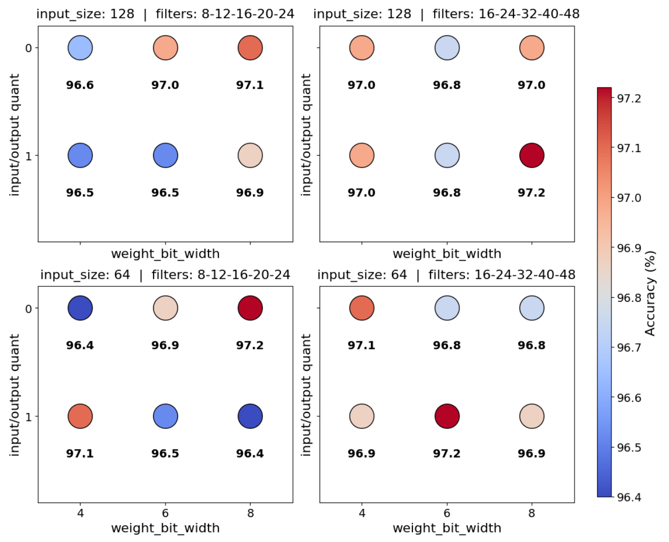

Fig. 2. Classification – Accuracy.

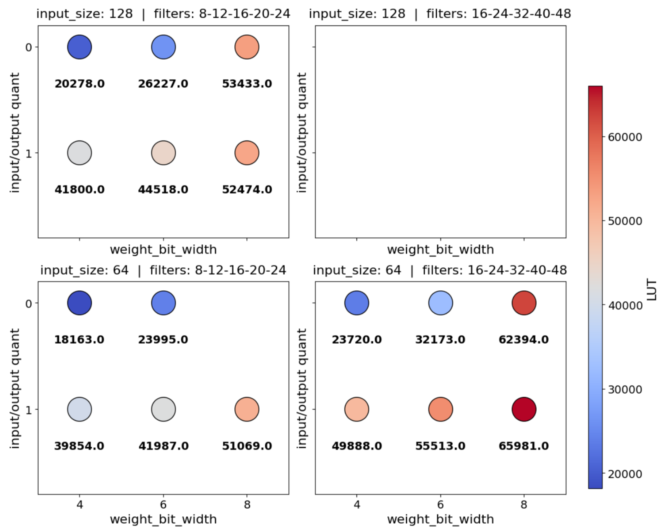

Fig. 4. Classification – LUT usage across input sizes, filter counts, and quantization settings.

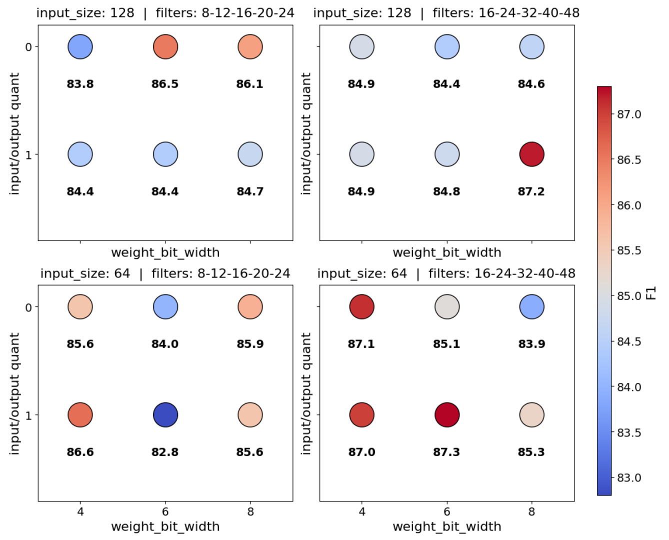

Fig. 3. Classification – F1.

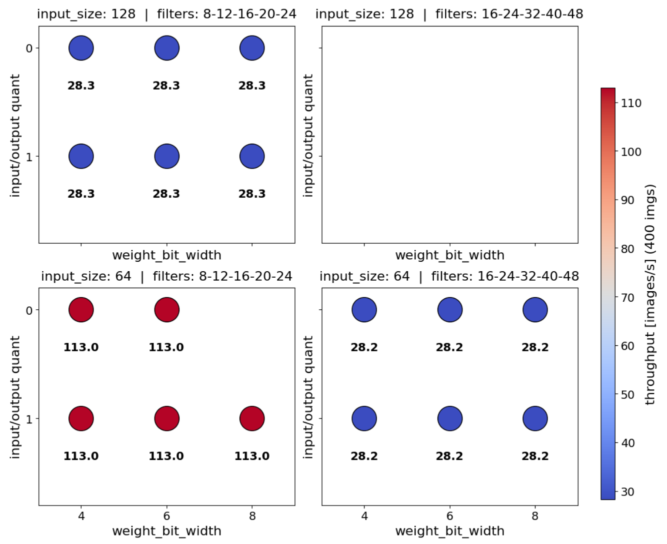

Fig. 5. Classification – Throughput [images/s] across different configurations.

and bit-widths significantly degrade error. In particular, using 2-bit weights resulted in catastrophic performance with average errors from 150–500 µm. Moderate configurations (e.g., 128×128, 6–8 bit) showed better stability. *See Figure 6.*

LUT Usage: Regression follows a similar pattern to classification: larger inputs, filters, and higher bit-widths result in steep LUT increases, often surpassing FPGA limits. *See Figure 7.*

Throughput: Throughput trends mirror classification: smaller inputs and fewer filters yield higher speeds (>110 images/s). Again, bit-width has negligible effect on speed. *See Figure 8.*

3) *Configuration Comparison*

To identify optimal configurations under multiple constraints, we visualized the regression landscape with respect to LUT utilization, average prediction error, and throughput. Based on this analysis, the two most effective configurations were selected by respecting the LUT constraint and optimizing the trade-off between average error and throughput.

See Figure 9.

Best Configurations:

- **Configuration A:**
  input_size = 64, weight bits = 8, filters = 8-12-16-20-24, no quantized I/O
  avg err = 34.77 µm, LUT = 52739, throughput = 112.96 images/s
- **Configuration B:**
  input_size = 64, weight bits = 6, filters = 16-24-32-40-48, no quantized I/O
  avg err = 23,98 µm, LUT = 32501, throughput = 28.24

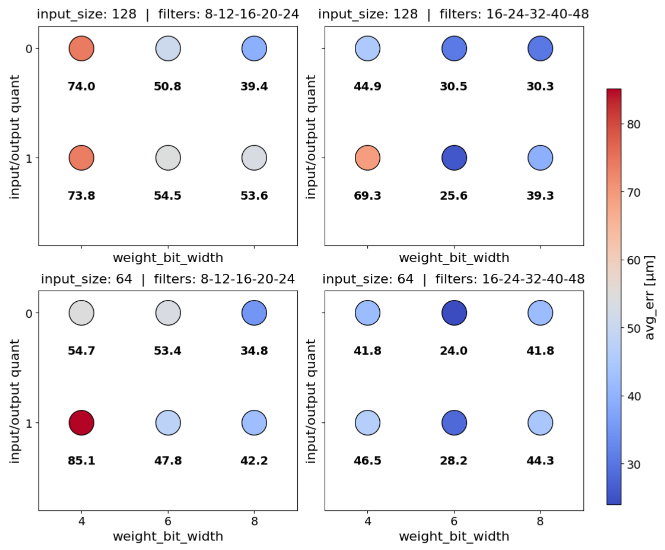

Fig. 6. Regression – Average error [µm] across bit-widths and quantization.

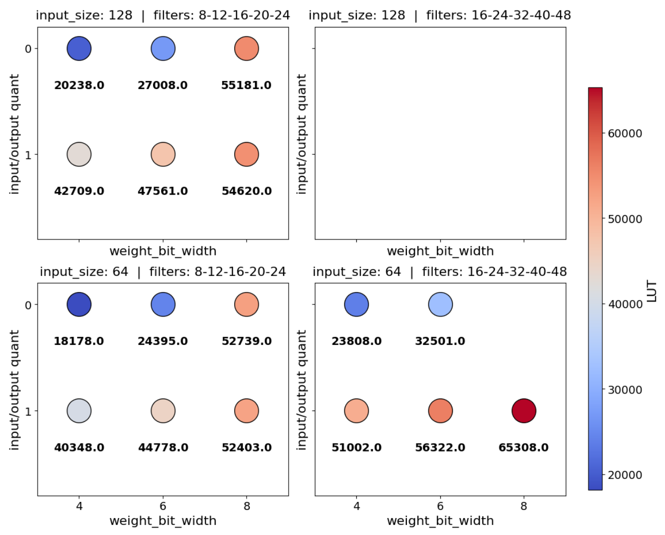

Fig. 7. Regression – LUT usage across input sizes, bit-widths, and quantization.

images/s

Configuration A achieves the lowest average error and operates in the highest throughput regime, while still satisfying the hardware resource constraints. In contrast, Configuration B provides an even lower average error but at the cost of reduced throughput—remaining nonetheless within the LUT constraint.

Input Parameter Effects (Key Trends):

- Increasing the **number of filters** consistently improves both classification accuracy and regression precision. However, this improvement comes at the cost of significantly higher LUT and FIFO usage. Past a certain threshold, the configurations become infeasible on Ultra96v2.
- **Bit-widths** show a clear trend: increasing the weight bit-width from 1 to 8 leads to a monotonic improvement in regression accuracy (i.e., lower average error), with diminishing returns beyond 6 bits. However, this also increases LUT usage following an almost trend. Very low bit-widths (1–2) cause a sharp degradation in performance, making them unsuitable for regression. Thus, 4–6 bits offer a practical balance between precision and hardware cost.
- **I/O quantization** provides a resource-saving mechanism, noticeably reducing LUT usage (especially in models with large inputs or many filters), while having a limited impact on accuracy—particularly for classification. It becomes a valuable tool to push borderline configurations within deployable bounds.
- **Input resolution** has a dominant influence on regression performance. Higher resolutions consistently lower average error, especially in moderate to high quantization regimes. Conversely, classification tasks are less sensitive to resolution changes, maintaining strong accuracy even at 64×64.

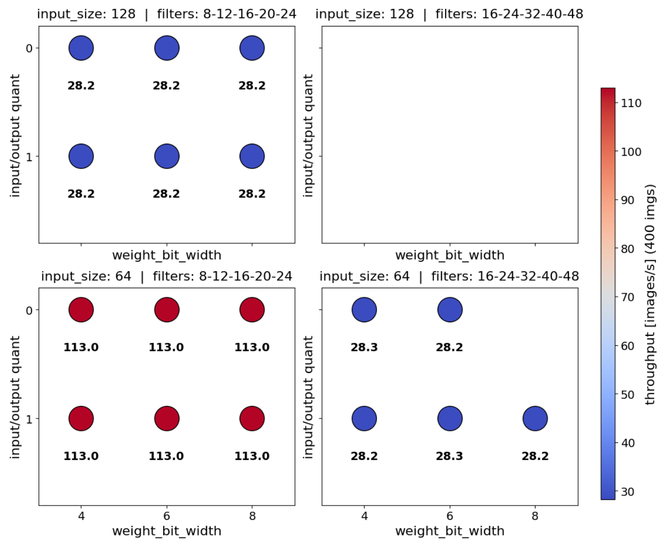

Fig. 8. Regression – Inference throughput [images/s] for various configurations.

Final Variable Sensitivity (Critical for Deployment Decisions):

- **Average error and LUT usage** are influenced by a complex interplay of multiple parameters — including input size, filter count, weight bit-width, and I/O quantization. Extensive additional testing confirmed that there is no single parameter whose adjustment alone leads to dramatically improved results. Instead, it is the careful combination and joint tuning of all these architectural choices that enables optimal trade-offs between accuracy and hardware feasibility. This reinforces the need for holistic design strategies rather than isolated optimizations.
- **Throughput** is largely independent of quantization and is

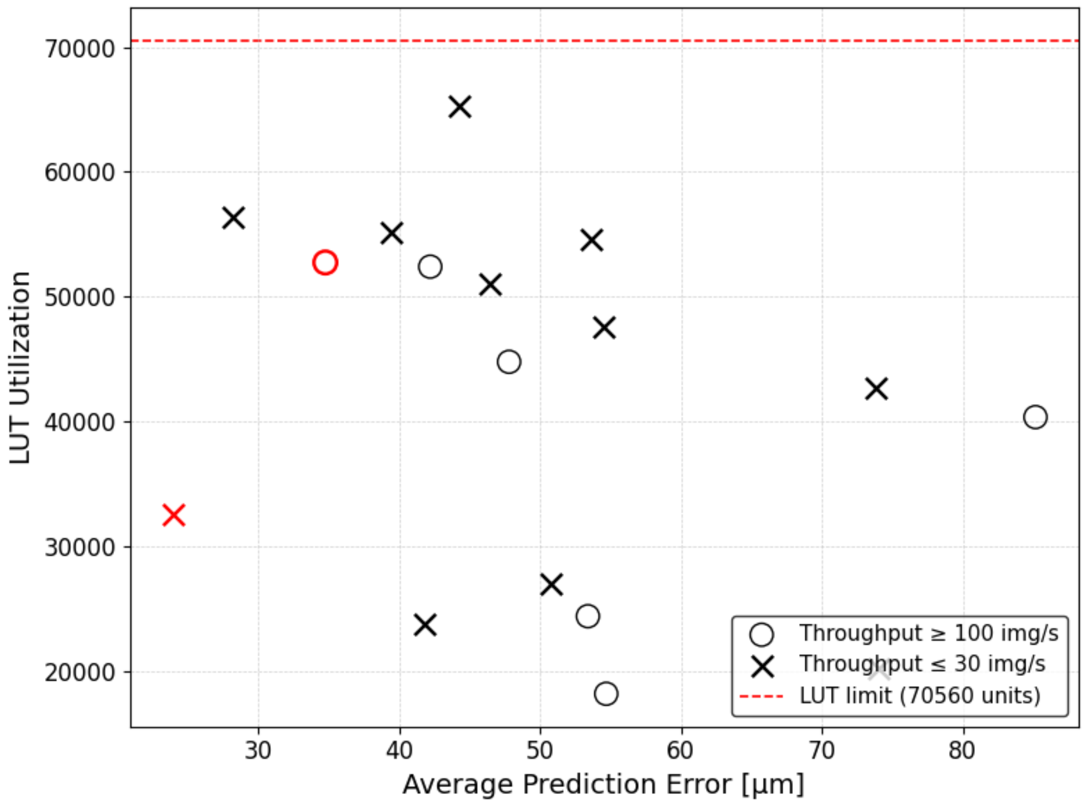

Fig. 9. Combined evaluation: LUT vs. average error [µm], colored by overall score. The red dashed line shows the Ultra96v2 LUT threshold.

governed almost exclusively by input resolution and filter count. This makes it easier to control independently, but it still interacts with resource constraints, which are in turn affected by quantization.

V. Conclusions

This work presented a hardware-aware design and evaluation pipeline for deploying quantized Convolutional Neural Networks (CNNs) on resource-constrained FPGA platforms. The classification models demonstrated strong robustness to quantization and architectural simplifications, achieving high accuracy across a range of compact configurations. In contrast, the regression models required more careful tuning of input resolution, bit-widths, and filter complexity to balance predictive precision and feasibility.

A central focus of the study was on navigating the trade-offs between average prediction error, inference throughput, and hardware resource utilization. Crucially, LUT usage was treated as a strict constraint imposed by the hardware limits of the Ultra96v2 platform. This constraint dictated the feasible region of the design space and guided the architectural choices throughout the quantization and deployment pipeline.

Through hardware parameter tuning, the models were able to balance and identify an optimal configuration that satisfied the LUT usage constraint while minimizing average error and maximizing throughput.

Overall, this work highlights the necessity of joint architectural and hardware-aware optimization when targeting embedded FPGAs.

Future improvements include enhancing the synthesis process for better adaptability to constrained platforms (improved FIFO estimation in FINN) and investigating fine-grained folding strategies to further improve resource efficiency and model scalability.

References

[1] J. Delon, Y. Gousseau, and A. B. Hamza, “A Short Introduction to Convolutional Neural Networks,”

[2] D. Chawda and B. Senouci, ”Fast prototyping of Quantized neural networks on an FPGA edge computing device with Brevitas and FINN,”

[3] Q. Ducasse, P. Cotret, L. Lagadec, and R. Stewart, “Benchmarking Quantized Neural Networks on FPGAs with FINN,”

[4] Z. Li, F. Z. Hong, and C. P. Yue, “FPGA-based Acceleration of Neural Network for Image Classification using Vitis AI,”