2025 FPGA 智慧運算與終端節點創意應用競賽
A3D3 Special Track 初賽報告書
組員: 陳冠晰、李昀達

1. 摘要與動機 Abstract and Motivation

1.1. Background and Approach

In recent years, the integration of machine learning (ML) into real-time scientific and embedded systems has become increasingly important. Areas of research such as high-energy physics, neuroscience, and astrophysics require low latency and high-throughput ML inference to process. With hardware-level customization, FPGAs can support inference in the microsecond range compared to traditional models and hardware which fail to meet these strict constraints. High-Level Synthesis (HLS) tools like hls4ml enables automated conversion of Keras-trained models to synthesizable C++ for FPGAs, accelerating hardware/software co-design and lowering the threshold for ML model deployment.

This challenge focuses on efficiently deploying quantized neural networks on FPGAs using hls4ml under strict constraints: under 75% utilization of all FPGA resources (LUT, FF, BRAM, DSP), latency below 33.3ms, and at least 60% accuracy on a CIFAR-10-based dataset. To meet these requirements, models were trained using Quantization-Aware Training (QAT) via QKeras, which simulates quantization effects during training to maintain model accuracy under low-precision constraints. hls4ml (version 1.1.0) is then used to generate an HLS-compatible configuration that was synthesized and deployed to FPGA hardware using the Xilinx Vitis HLS 2023.2 toolchain, targeting the PYNQ-Z2 development board.

1.2. Result Overview

Our final model is composed of four convolutional layers and two dense layers. It maintained strong performance with 67.38% test accuracy and 62.8% accuracy after HLS conversion, meeting requirements despite aggressive quantization. The model also achieved a latency of 12.57ms and passed hardware resource constraints with 46% LUT, 33% FF, 66% BRAM, and 21% DSP utilization, falling well below the resource usage limit. The result demonstrates a well-balanced trade-off between model accuracy and hardware deployment, showcasing the viability of QAT and HLS-based deployment for edge ML applications.

2. 設計方法與量化技術 Design Methodology and Techniques

2.1. Model Architecture

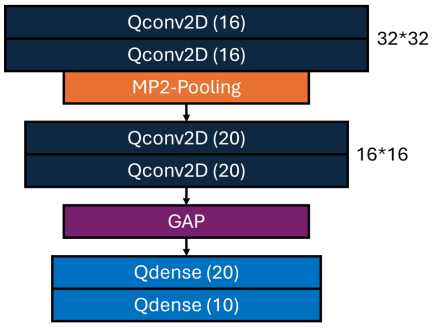


Fig. Model Architecture

Our model begins with an input layer for 32×32×3 images, followed by two convolutional stages. The first stage consists of two convolutional layers with 16 filters each, followed by quantized ReLU activations and a Max Pooling layer to reduce spatial resolution and computation. The second stage includes two convolutional layers with 20 filters each and a GlobalAveragePooling layer to reduce feature dimensionality connecting to dense layers without introducing additional parameters. All layers use the QConv2DBatchnorm module, which fuses batch normalization into the convolution for optimized inference and reduced latency. The final classification block consists of two fully connected QDense layers: the first layer has 20 units followed by batch normalization, quantized ReLU, and dropout, while the second outputs 10 class scores. Dropout is applied after each major stage (rates of 0.1, 0.2, and 0.3) for regularization, and L1/L2 penalties are applied to all convolutional and dense layers to encourage sparsity and stability.

2.2. Data Augmentation

Since a diffusion model-generated image dataset is used as the final assessment for model accuracy, enhancing the generalization and robustness of the model was crucial. Thus, a diverse set of data augmentation techniques were used during preprocessing.

- CutMix combines two randomly selected images by replacing a patch of one image with a region from another, and blending their labels proportionally, encouraging the model to learn from mixed spatial and contextual features.
- Gaussian blur was applied with a 30% probability using 3×3 or 5×5 kernels to reduce sensitivity to high-frequency artifacts.
- Color jitter was introduced with a 50% chance by randomly brightening a single RGB channel, mimicking illumination variability.
- Gaussian noise with a standard deviation of 0.06 was also added to each image to simulate sensor or environmental noise, adding resilience to input variations.
- Random erasing was applied with a 50% chance by masking a random region of 1/8–1/4 of the image size, simulating occlusions to enhance spatial robustness. (used as alternative to CutMix)

In total, 100,000 augmented samples were generated and combined with the original dataset, effectively doubling its size.

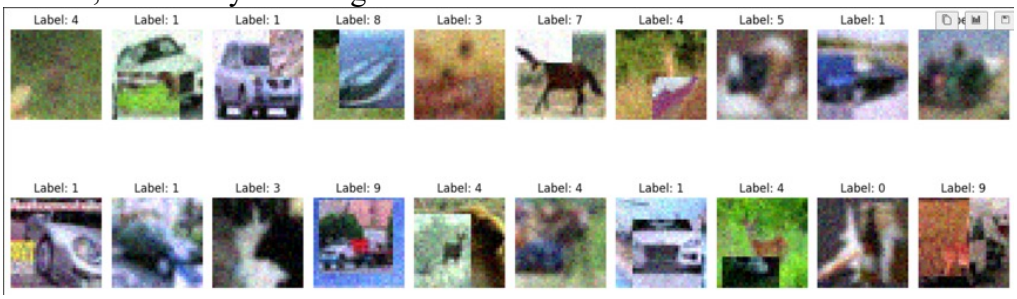


Fig. Augmented images examples

2.3. Model Training

For stable convergence and high generalization performance, the model training process includes a warm-up phase followed by cosine learning rate decay. During the first five epochs, the learning rate gradually increases from zero to the base rate of

1×10^{-3}, providing a stable start to training. After the warm-up phase, the learning rate follows a cosine decay schedule, slowly decreasing to a minimum value of 1×10^{-7} across 150 total epochs. The training batch size is set to 32 to balance training time and generalization to data. Two Keras callbacks are used: ModelCheckpoint monitors validation accuracy and saves the best-performing model, while LearningRateScheduler dynamically adjusts the learning rate each epoch based on the defined schedule. These strategies avoid abrupt changes in optimization and ensure that training is both efficient and robust, yielding a quantized model that performs well while maintaining highly deployable on FPGA platforms.

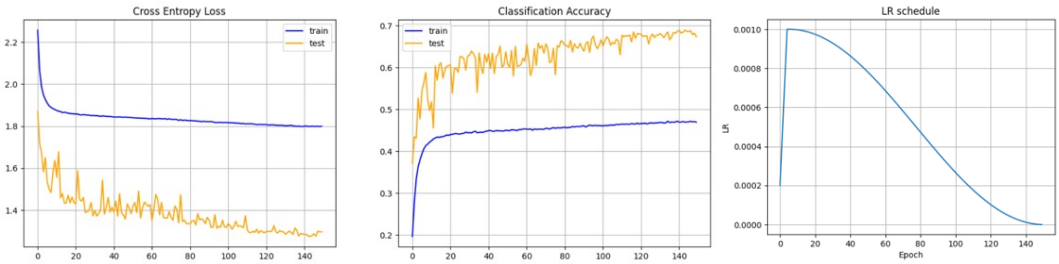


Fig. Loss, accuracy under learning rate with warm-up and cosine decay

2.4. Quantization

The weights and biases in all layers were quantized using the format quantized_bits(8, 2, alpha=1). Using 8-bit fixed-point with 2 integer bits and trainable scaling (alpha=1) provides a good trade-off between numerical precision and hardware efficiency. QActivation('quantized_relu(8,2)') was also consistently applied after each convolutional and dense block.This clips activation outputs to the same fixed-point range, maintaining uniformity across the model. Additionally, input layers use an unsigned format ufixed<8,0>, while batch normalization layers operate with a more precise fixed<18,8> representation, giving enough resolution for statistical stability without excessive resource usage. This QAT setup ensures accurate inference under limited bit widths and contributes to significantly lower BRAM and DSP utilization in the FPGA implementation.

2.5. Reuse Factor

Given that the latency constraint for inference is relatively lenient, hardware utilization on the FPGA became the primary bottleneck in the design. Additionally, RTL simulation deadlocks are a common issue when layer throughput is mismatched. To avoid this, reuse factors were assigned individually to each layer, allowing movement between layers without stalling or requiring excessive buffering.

For example, earlier convolutional layers were given higher reuse factors (e.g., 216 to 288), as these layers process larger feature maps and benefit more from reduced parallel computation. Similarly, both dense layers were set to a reuse factor of 200 to balance resource savings with acceptable latency. The HLS strategy was set to "Resource" to explicitly prioritize reducing logic and memory usage over latency.

3. 實驗結果討論 Experimental Results and Discussion

3.1. Model Size

The final quantized model contains a total of 10,290 parameters, of which 10,102 are trainable and 188 are non-trainable (primarily from batch normalization layers). This

compact size makes our design highly suitable for FPGA deployment while still allowing enough depth to learn trends and features through our dataset. While the parameter count reflects the compactness of the model, the true impact on hardware is better captured through post-synthesis FPGA resource utilization.

3.2. FPGA Resource Utilization

LUT & FF:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	24503	0	0	53200	46.06
LUT as Logic	22719	0	0	53200	42.70
LUT as Memory	1784	0	0	17400	10.25
LUT as Distributed RAM	0	0			
LUT as Shift Register	1784	0			
Slice Registers	34953	0	0	106400	32.85
Register as Flip Flop	34953	0	0	106400	32.85
Register as Latch	0	0	0	106400	0.00
F7 Muxes	517	0	0	26600	1.94
F8 Muxes	232	0	0	13300	1.74

BRAM:

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	92.5	0	0	140	66.07
RAMB36/FIFO*	75	0	0	140	53.57
RAMB36E1 only	75				
RAMB18	35	0	0	280	12.50
RAMB18E1 only	35				

DSP:

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	46	0	0	220	20.91
DSP48E1 only	46				

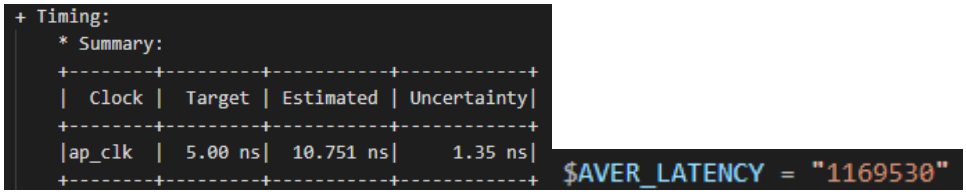
Our final implementation achieved successful synthesis and deployment on the target FPGA platform while keeping all resource usage well below the competition thresholds. Specifically, the resource utilization for the current configuration was as follows:

- LUTs: 46.06%
- Flip-Flops (FFs): 32.85%
- Block RAM (BRAM): 66.07%
- DSPs: 20.91%

Although this configuration met all hardware constraints, the design process encountered significant challenges. Specifically, the HLS toolchain frequently stalled at the RTL simulation stage due to suspected memory bottlenecks. To resolve this, we migrated our project to a workstation with Xeon(R) Gold 6326 CPU with 512 GB of RAM, located in our university lab. On this system, our final model configuration completed RTL simulation without deadlock. Unfortunately, due to limited access to this workstation (only two days before submission), we were unable to re-evaluate previously failed configurations. Nonetheless, the selected configuration satisfied both the resource and simulation requirements, enabling a valid final submission.

3.3. Latency Analysis

Based on the Vivado HLS report:



- Estimated Clock Period: 10.751 ns
- Average Cycle Count (from RTL simulation): 1169530 cycles

This results in a total inference latency of approximately: 12.57ms
(latency = Csynth estimated clock period * RTL-sim average cycle count)
This is well within the timing constraints, demonstrating that the model is suitable for deployment even when prioritizing hardware resource utilization.

3.4. Accuracy and Tradeoff

Our final model achieved:

- Training Accuracy: 47.19%
- Validation Accuracy: 68.95%
- Testing Accuracy: 67.38%
- Kaggle (Post-HLS) Accuracy: 62.8%

The lower training accuracy is attributed to our aggressive data augmentation strategy, which prioritizes generalization over fitting the training set. The final quantized HLS model preserved much of the learned representation, though the slight drop in accuracy on the post-HLS version may be due to additional fixed-point rounding and quantization effects introduced during hardware mapping.

3.5. Discussion

Throughout the development process, we explored various architectures to balance accuracy, parameter count, and resource utilization. In one instance, a larger model with ~35,000 parameters achieved a promising Kaggle test accuracy of 76.8%, but the excessive size prevented successful RTL simulation due to resource and memory overflows. Thus, we focused more on smaller models with 4~6 convolution layers.

We also noticed that 6-layer CNNs consistently failed to meet the FPGA resource requirements, despite having a smaller parameter count (7000~8000), particularly exceeding the limits for BRAM and DSP utilization. This is because deeper networks inherently require more intermediate feature maps, and arithmetic operations, especially in convolutional layers. As the number of layers increases, the cumulative demand for memory to store weights, activations, and intermediate outputs grows significantly. It also increases the chance of speed mismatch when tuning individual reuse factors of layers. As a result, we concluded that a shallower 4-layer CNN strikes a better balance between model complexity and hardware feasibility under the constraints of the target FPGA.

We believe that with additional training epochs, the model's performance could be further improved, based on the continuing improvement trend observed in the loss and accuracy curves. However, our progress was constrained by limited hardware availability and time. In future work, we plan to extend training and conduct broader experiments with more complex models, leveraging more powerful hardware setups to fully explore the performance-resource trade-off.