Final Report: Weight-Stationary 8x8 Systolic Array for VGG16 Quantization-Aware Training

Ashish Murthy ● Ben Zhang ● Hojin Chang ● Karan Humpal ● Nithin Baimeedi

Introduction

Deep convolutional neural networks (CNNs) excel at vision tasks but are costly in energy, memory, and computation. These limitations hinder their deployment on embedded and edge devices. We present a weight-stationary, 8x8 systolic array accelerator for VGG16 with quantization-aware training and targeted compression to address these constraints.

Our approach integrates structured and unstructured pruning to remove redundant weights and applies Huffman encoding to reduce off-chip data transfers. This design is implemented on the Cyclone IV GX FPGA and significantly cuts computational overhead and energy use while maintaining state-of-the-art accuracy.

We deliver efficient, high-performance CNN inference suitable for space, power, and cost-constrained environments by uniting pruning, quantization, and compression.

Part 1: Quantization-Aware Training

- We trained a modified VGG16 model with quantization-aware training using 4-bit input activations and weights. The 27th convolutional layer was reduced to 8 input and 8 output channels, and batch normalization was removed for the modified layer. The computed psum_recovered values, including ReLU, were compared with precooked inputs for the next layer.

Results

- Achieved validation accuracy of 90.79% with the modified VGG16 model.
- Verified psum_recovered values with an error of 8.2 × 10^-4, meeting the required threshold of 10^-3.

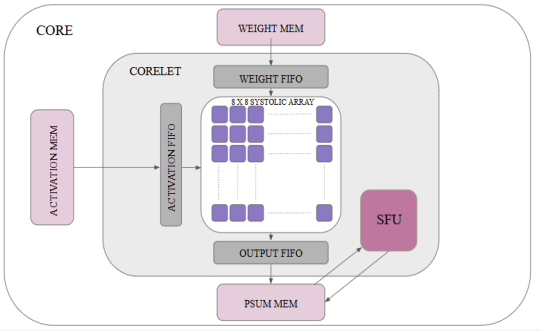
Conclusions

- Reducing the channels for the selected convolution layer and removing batch normalization allowed efficient mapping onto the systolic array without tiling while maintaining high accuracy. Quantization-aware training ensured robust results even with 4-bit quantization.

Part 2: RTL Core Design

Overview

- We designed and connected components, including the 8x8 systolic array with MAC units, scratchpad memories for activations, weights, and psums, L0 buffer, output FIFO, and a Special Function Processor (SFP) for accumulation and ReLU. The corelet.v module integrated all components except SRAMs.



Results

- Successfully connected all components without compilation errors.

Conclusions

- The modular design ensured seamless integration of all components and facilitated FPGA mapping. The successful compilation validated the functional design.

Part 3: Testbench Generation

Overview

- We created a testbench to verify the squeezed layer (8x8 channels). The testbench performed input SRAM loading, kernel data loading to PE registers, execution in the PE array, psum movement, accumulation, ReLU, and output file generation. We compared the results with the expected outputs.

Results

- Stimulus (input.txt, weight.txt) and expected output (output.txt) files were generated.
- Achieved zero verification errors when comparing RTL outputs to expected results.

Conclusions

- The testbench verified all stages of the computation pipeline with high reliability. Successful verification demonstrated the correctness of the design.

Part 4: FPGA Mapping

Overview

The corelet.v module was prepared for mapping to the Cyclone IV GX FPGA using Quartus Prime 19.1. Synthesis, placement, and routing were performed, and performance metrics, including frequency, power, and TOPS-related benchmarks, will be measured.

Part 5: Reconfigurable PE Design

Overview

- A reconfigurable PE was designed to support weight and output stationary modes. Mux-based routing shared input, weight, and output registers across modes. The design included an IFIFO for weight data and verified functionality using the first convolutional layer's input, weight, and activations.

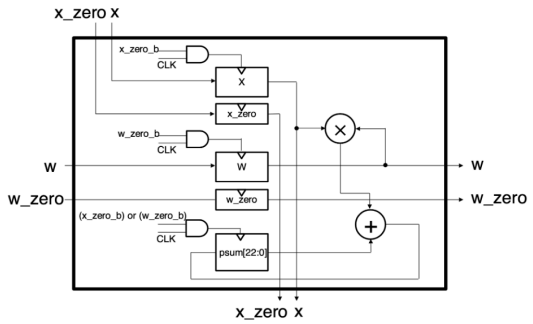
Results

- Achieved zero verification errors when comparing RTL outputs to PyTorch simulation results.

Conclusions

- The reconfigurable PE provided flexibility for different workloads, maintaining functional accuracy and efficient data flow management.

Part 6: +Alpha Enhancements
Overview
The project included several +alpha enhancements to improve functionality and efficiency:

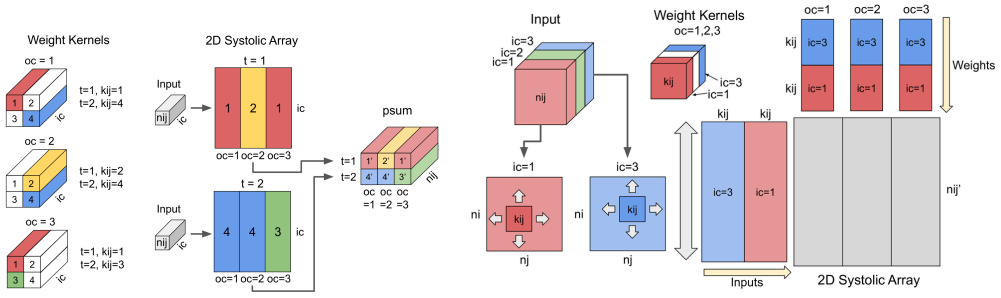
1. **Unstructured Pruning - Gating:**
    a. Unstructured pruning removes individual weights within each layer, increasing sparsity by setting specific parameters to zero. Although it achieves high sparsity, it generates irregular patterns that can complicate hardware implementation. We eliminate redundant weights, allowing the hardware to bypass computations associated with these pruned elements.
    b. 

2. **Huffman Encoding:**
    a. Applied Huffman encoding to compress weight data, reducing the memory footprint significantly.
    b. Encoding helped maximize on-chip storage, enabling faster access times and reducing the reliance on external memory resources.
    c.

Most efficient		Implemented	
0000	0	0000	0
0001	01	0001	01
0010	001	0010	001
0011	000	0011	0001

3. **Weights Stationary Time Skipping (Structured Pruning):**
    a. Structured pruning optimizes computations on a systolic array by using sparsity patterns in convolutional filters. It eliminates sets of weights, like specific input channels, leaving only essential values, thereby reducing data movement and arithmetic operations. A weight-stationary approach keeps unpruned weights fixed, avoiding reloading. Focusing on output channels enables the hardware to skip computations for pruned inputs, improving speed in line with sparsity levels. Despite reducing parameters and quantization, structured pruning maintains competitive accuracy, enhancing efficiency in constrained hardware environments.

b.



Alpha Results

- **Unstructured Pruning - Gating:**

Power Consumption using $P = CV^2 f$	W. S.
No Hardware Gating	30.90mW
After Hardware Gating	11.64mW

- **Output Stationary Time Skipping (Structured Pruning):**

Accuracy of VGG16 on CIFAR10	W. S.	O. S.
Unpruned Full Precision	90.98%	
78% Sparse Full Precision	90.77%	86.68%
78% Sparse 4bit Quantized	90.79%	87.36%

- **Huffman Encoding:**

Data	Compression Ratio
Activation	2.49
Weights	1.16

Alpha Conclusions

- We achieve significant efficiency gains without sacrificing accuracy by integrating unstructured pruning, Huffman encoding, and output stationary time skipping (structured pruning).. Unstructured pruning with gating removes individual weights, cutting unnecessary computations and power use. Huffman encoding compresses weights, reducing memory reliance and access times. Structured pruning streamlines the systolic array's workload, translating weight sparsity directly into speedups. These techniques maintain competitive accuracy for quantized VGG16, enabling high-performance, energy-efficient deep learning on resource-constrained platforms.

Summary Table

Metric	OPs	Frequency	Dynamic Power	GOPs / s	GOPs / W	Logic Elements
VGG16	128	118.84MHz	30.90mW	15.21	0.00492	17368

Conclusion

We created a weight-stationary 8x8 systolic array accelerator tailored for a quantized VGG16 model, achieving above 90% accuracy despite notably lower data precision and hardware complexity. By employing quantization-aware training, reducing channels, and eliminating batch normalization, we enabled efficient mapping to the systolic array without tiling. Extensive testbench verification confirmed functional correctness.

Additional improvements, such as structured and unstructured pruning and Huffman encoding, have lowered power consumption, memory usage, and computational demands while maintaining accuracy.

In the future, we can investigate more rigorous pruning strategies, enhanced quantization approaches, and sophisticated compression methods to improve efficiency further and adapt to increasingly limited hardware resources.