# ECE 284 Final Project Report

SIMD + Reconfigurable Systolic Array Accelerator

Sujen Kancherla, Amaan Mohammed, Matteo Persiani, Rishi Pothukuchi, Daniel Sanei
UC San Diego

December 14, 2025

## Contents

# 1 Part 1: Vanilla Version (4-bit)

## 1.1 Model Training

We trained a VGG16 model with 4-bit quantization for both weights and activations on the CIFAR-10 dataset. We reduced both input and output channels to 8 for the `features[27]` convolutional layer (Conv2d with 3×3 kernel) and removed the batch normalization layer after it as well.

## 1.2 Hardware Design

- Components: 8×8 systolic array with 4-bit MACs, weight-stationary dataflow, Input/Output SRAM, L0/OFIFO, Special Function Unit

The vanilla accelerator implements an 8×8 weight-stationary systolic array with 64 processing elements (PEs). Each multiply-accumulate (MAC) performs a 4-bit × 4-bit operation with zero-extended activation values, producing 16-bit outputs. The weights are loaded in a wave-front pattern, and the PE tiles contain activation, weight, and partial sum registers. The systolic array propagates instructions diagonally, accepting 8 parallel activations from the west and outputting 8 partial sums to the south.

The memory hierarchy includes a 32-bit Input SRAM, 128-bit Output SRAM, L0 FIFO for buffering, and OFIFO for output collection. The SFU performs 8-lane accumulation and ReLU in bypass, accumulate, or finalize modes. A 35-bit instruction bundle at the testbench level controls all operations.

## 1.3 Verification & Conclusion

**Results:** The trained model achieved **90.33% accuracy** on the test set, exceeding the required 90% threshold. For the target layer, we captured the input to `features[27]` (shape: [128, 8, 4, 4]) and verified it by comparing our computed `psum_recovered` (after ReLU) against the prehooked input to `features[29]`. The verification showed a mean error of $3.78 \times 10^{-7}$ and maximum error of $7.87 \times 10^{-6}$. Additionally, when compiling and verifying the hardware implementation using our Verilog testbench, we were able to successfully compute values that matched the expected results.

## 1.4 FPGA Synthesis Results

| Metric | Value |
|---|---|
| Frequency | 100 MHz |
| Fmax (slow corner, 40 °C) | 118.71 MHz |
| Dynamic Power | 209.62 mW |
| Peak Throughput | 12.8 GOPS/s |
| Energy Efficiency | 61.0 GOPS/W |
| LUTs / Registers | 17,348 / 12,218 |

# 2 Part 2: 2-bit/4-bit Lane Reconfigurable SIMD Array

## 2.1 Model Training

We trained a VGG16 model with mixed-precision quantization: 4-bit weights and 2-bit activations. The key architectural change was expanding the target layer (`features[27]`) from 8 to 16

input/output channels, a 2×2 tiling scheme to map onto the 8×8 systolic array. We still removed the batch norm after this layer as well.

### 2.2 Design Modifications

- Components: 8×8 array with reconfigurable SIMD MACs, dual-weight registers per PE, 36-bit instruction bundle

This part extends the vanilla design with Single Instruction, Multiple Data (SIMD) reconfigurability to support both 4-bit (8 input channels) or 2-bit (16 input channels) activation values. This essentially doubles the throughput for lower precision operations.

The MAC supports the following mode-controlled operations: a standard 4-bit x 4-bit (when mode = 0) or a dual 2-bit x 4-bit SIMD (when mode = 1). In 2-bit mode, the activation values split into upper and lower halves, and are each multiplied by separate weights. The PE tiles require a second weight register and dual-cycle loading for the 2-bit mode, and the mode signal is broadcast to all PEs, thereby requiring the instruction bundle to expand to 36 bits.

### 2.3 Verification & Conclusion

**Results:** The model achieved **91.35% accuracy** on CIFAR-10, exceeding both the 90% threshold and Part 1's performance. The 16-channel layer (shape: [128, 16, 4, 4]) required tiling into 2 input channel tiles and 2 output channel tiles, with 4 weight tiles total. The verification showed a mean error of $1.0 \times 10^{-6}$ and maximum error of $1.9 \times 10^{-5}$. The hardware design for this section was completed, however the testbench is incomplete.

## 3 Part 3: Weight/Output Stationary Reconfigurable PE

### 3.1 Design Modifications

- Components: 8×8 array with dataflow muxes, L0 + IFIFO, 36-bit instruction bundle

This part extends the vanilla design to implement dataflow reconfigurability, which allows the systolic array to switch between Weight Stationary (WS) and Output Stationary (OS) modes. This enables mapping different architectures efficiently on the same hardware by changing how data flows through the systolic array. For WS mode (when mode = 0), the weights are stationary, activations flow from west to east, and partial sums flow from north to south. For OS mode (when mode = 1), the weights flow from west to east, activations flow from north to south, and partial sums accumulate while stationary. The PE tiles include new muxes that reroute the signals while sharing the registers, and a new IFIFO (8 parallel FIFOs with a depth of 64) is created to buffer the weights for the OS mode. WS mode uses L0 for the activations, and OS uses IFIFO for the weights and L0 for the activations. The instruction bundle is expanded to 36 bits to include the new mode signal as well.

### 3.2 Verification & Conclusion

The hardware design for this section was completed, however the testbench is incomplete for the overall output stationary flow.

# 4 Part 4: Alpha

## 4.1 Alpha 1: Leaky ReLU

Replacing ReLU with Leaky ReLU in our quantized VGG16 improved classification accuracy by +0.41% on CIFAR-10. This gain is likely due to improved gradient flow and reduced saturation in 8-channel layers.

The Leaky ReLU activation function is defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{1}$$

where $\alpha$ is a small positive constant. In our implementation, we used $\alpha = 0.5$. This allows a small gradient to flow through negative inputs, preventing the "dying ReLU" problem where neurons can become permanently inactive.

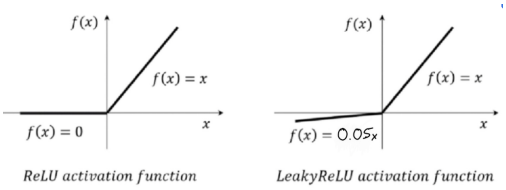


Figure 1: Leaky ReLU activation function comparison

### 4.1.1 Hardware Impact

The low overhead (minor comparator/shifter logic) is negligible compared to the MAC datapath and memory, resulting in no expected change to frequency, throughput, or power consumption.

## 4.2 Alpha 2: Cosine Annealing Learning Rate

Implementing Cosine Annealing with a 10-epoch warm-up improved convergence stability and overall accuracy by +0.82% on CIFAR-10.

The Cosine Annealing learning rate schedule is defined as:

$$\eta(t) = \eta_{\min} + (\eta_{\max} - \eta_{\min}) \cdot \frac{1 + \cos(\pi \cdot t/T)}{2} \tag{2}$$

where $t$ is the current epoch, $T$ is the total number of epochs, $\eta_{\max}$ is the initial learning rate, and $\eta_{\min}$ is the minimum learning rate. The learning rate smoothly decreases from $\eta_{\max}$ to $\eta_{\min}$ following a cosine curve, which helps the model converge more smoothly to a better local minimum.

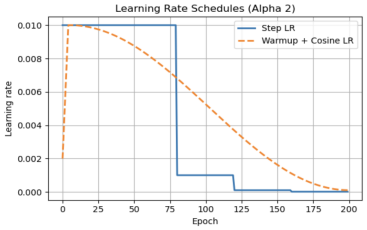


Figure 2: Cosine annealing learning rate schedule vs step learning rate scheduler

#### 4.2.1 Hardware Impact

Cosine annealing only changes the training schedule. The deployed fixed-point model and accelerator hardware remain identical, so throughput, power, and area are unchanged.

### 4.3 Alpha 3: Activation-Aware Pruning

Traditional pruning uses magnitude of the weights to prune them. We implemented activation-aware pruning where we used sample images from training data to take the activation and prune weights based on the activation. Our hope was to prune the least contributing weights as well as the least important weights in magnitude also.

#### 4.3.1 Methodology

Three pruning strategies were implemented and compared: traditional magnitude pruning, activation-aware pruning, and a hybrid approach. The activation-aware method utilizes an `ActivationCollector` module that registers forward pre-hooks on quantized convolutional layers to capture per-channel activation statistics. For each input channel, the mean absolute activation is computed as $\bar{a}_c = \mathbb{E}[|a_c|]$.

The importance metric for activation-aware pruning is defined as $I_{ij} = |w_{ij}| \times \bar{a}_i$, where $w_{ij}$ denotes the weight connecting input channel $i$ to output channel $j$. The hybrid approach combines normalized magnitude and activation scores: $I_{hybrid} = \alpha \cdot I_{mag}^{norm} + (1 - \alpha) \cdot I_{act}^{norm}$. Pruning is performed globally by computing importance thresholds, generating binary masks, and zeroing out parameters below the threshold.

#### 4.3.2 Results

Table 1: Accuracy comparison across sparsity levels for three pruning methods

| **Sparsity** | **Pruning** | **Act-Aware** | **Hybrid** |
|---|---|---|---|
| 30% | 90.37% | 90.25% | 90.11% |
| 50% | 90.45% | 90.25% | 89.25% |
| 70% | 90.22% | 89.84% | 86.85% |

The activation aware pruning and hybrid approach actually had comparable accuracy on 30% and 50% sparsity.

Table 2: Hardware efficiency metrics at 50% sparsity

| **Metric** | **Pruning** | **Act-Aware** | **Improvement** |
|---|---|---|---|
| TOPS/Watt | 16.40 | 16.42 | +0.1% |
| PE utilization | 60.94% | 60.94% | — |
| Zero Clustering | 0.0405 | 0.0406 | +0.4% |

Activation-aware pruning demonstrated slightly superior hardware efficiency. At 50% sparsity, it achieved 16.42 TOPS/Watt compared to 16.40 for traditional pruning (+0.1% improvement). Layer-wise sparsity analysis revealed adaptive behavior, with early layers retaining 99.25% of parameters while deeper layers exhibited up to 64.80% sparsity, suggesting appropriate adjustment of pruning intensity based on layer importance.

### 4.4 Alpha 4: Gradual Channel Squeeze Architecture for Bottleneck Optimization

The original VGG16_quant architecture employs a bottleneck to make the hardware design simpler with the systolic array only computing 8x8 input to output channels. We implemented more layers before and after the compression bottleneck to smoothen the transition from 512 to 8. The goal is to increase performance with minimal hardware impact.

#### 4.4.1 Architecture Design

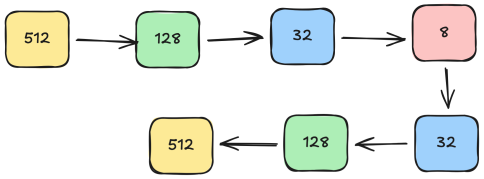


Figure 3: alpha 3 architecture

The VGG16_quant_gradual architecture replaces the single-step bottleneck with a symmetric eight-stage transition: $512 \rightarrow 128 \rightarrow 32 \rightarrow 8 \rightarrow 8 \rightarrow 32 \rightarrow 128 \rightarrow 512$. This introduces six additional quantized convolutional layers (features[24] through features[41]), each followed by batch normalization and ReLU activation.

The architecture maintains 4-bit quantization for weights and activations with learned scaling factors ($\alpha_w = 2.2114$, $\delta_w = 0.3159$ for weights; $\alpha_a = 5.7510$, $\delta_a = 0.3834$ for activations). Integer arithmetic verification confirms deployment feasibility on fixed-point hardware.

### 4.5 Results and Discussion

Table 3: Accuracy comparison across all model variants on CIFAR-10

| **Model** | **Accuracy** | **Improvement** |
|---|---|---|
| VGG16 (Vanilla) | 90.33% | — |
| Alpha 1 (Leaky ReLU) | 90.74% | +0.41% |
| Alpha 2 (Cosine Annealing) | 91.15% | +0.82% |
| Alpha 4 (Gradual Channel Squeeze) | 90.82% | +0.49% |

Table 3 summarizes the accuracy improvements achieved by each Alpha experiment. Alpha 2 (Cosine Annealing Learning Rate) achieved the highest accuracy improvement of +0.82%, followed by Alpha 4 (+0.49%) and Alpha 1 (+0.41%). All optimizations successfully improved upon the baseline vanilla model while maintaining hardware compatibility.