Project Report: Vectorized CNN Implementation on RISC-V

Muhammad Sharique Baig - 28369
Hamza Ahsan - 28903
Abdullah Faraz - 29271
Faraz - 29201

Methodology and Logic Behind Vectorization

The project aimed to implement a convolutional neural network (CNN) inference pipeline on a RISC-V processor utilizing its vector extensions for performance gains. Instead of training directly on the RISC-V platform, the approach involved importing pre-trained weights and biases from external files, then applying all CNN layers—convolution, ReLU activation, max pooling, dense layers, and softmax.

Vectorization focused on leveraging the RISC-V vector instruction set to process multiple data elements in parallel, maximizing throughput for large tensor operations fundamental to CNNs. Key algorithmic steps were adapted to:

- Load multiple input pixels and weights into vector registers simultaneously.
- Perform parallel multiply-accumulate operations for convolutions.
- Utilize vectorized instructions for activation and pooling operations.
- Manage data alignment and memory strides carefully to maintain vector efficiency.

This low-level optimization required a balance between algorithmic logic and hardware constraints, optimizing loop unrolling, register allocation, and memory access patterns for best performance.

Challenges and Resolutions

Debugging Vectorized Assembly Code

Debugging was the primary challenge throughout the project. Writing vectorized assembly for CNN operations demanded meticulous management of vector and scalar registers and careful use of vector instructions respecting length and masking constraints. The lack of sophisticated debugging tools at this level meant reliance on verbose logging, simulator traces, and step-by-step verification against Python implementations.

Register management posed a risk of accidental overwrites causing subtle bugs, necessitating rigorous code reviews and conservative coding patterns. Instruction set nuances further complicated implementation, requiring detailed understanding of RISC-V vector semantics.

Virtual Machine Stability

The RISC-V simulator environment frequently crashed or froze when executing complex vectorized code, hampering development flow. Crashes appeared linked to resource limits or immature vector instruction implementations within the simulator.

To mitigate this, the workflow was adapted to include incremental code testing with simplified scenarios, systematic use of execution logs for crash diagnostics, and fallback to scalar Python code for validating intermediate results. This hybrid approach enabled steady progress despite environmental instability.

Project Status and Remaining Work

Currently, all core CNN layers (convolution, ReLU, max pooling, dense, softmax) have been implemented in vectorized RISC-V assembly. The system successfully reads pre-trained model weights from files and processes the MNIST dataset for inference.

However, several tasks remain:

- **Performance Benchmarking:** Comprehensive profiling and optimization of vectorized code to measure and improve runtime efficiency.
- **Complete Integration:** End-to-end integration testing across all layers ensuring seamless data flow and correctness.
- **Real-World Dataset Testing:** Validation beyond MNIST with more complex datasets to assess generalization and robustness.
- **Training on RISC-V:** Exploring possibilities for on-device training or fine-tuning, currently absent.
- **Simulator and Toolchain Stability:** Continued efforts to stabilize the development environment and enhance debugging capabilities.