Implementation of CNN using RISC-V Vector Instructions for MINST Digit Classification

Group: Logic Loops
Aman Khan
Namrah Binte Abbas
Saadan Ashraf
Eesha Ali

Abstract—This project presents a complete implementation of a Convolutional Neural Network (CNN) forward pass using RISC-V vector instructions assembly language for MNIST digit classification. The implementation includes convolution operations, ReLU activation functions, fully connected layers, and prediction generation through argmax operations. The system processes 8×8 input images through multiple layers including convolution with 8 filters, ReLU activation, and two fully connected layers to produce digit classification results. The implementation demonstrates efficient utilization of RISC-V vector extensions for parallel processing of neural network computations, achieving computational efficiency through vectorized operations. The project successfully implements all core CNN components in low-level assembly, providing insights into hardware-accelerated machine learning inference at the instruction set architecture level.

Keywords—RISC-V, Vector Instructions, CNN, MNIST, Assembly

Machine learning inference on embedded systems and specialized hardware requires efficient low-level implementations that can maximize computational throughput while minimizing resource utilization. Convolutional Neural Networks (CNNs) have become the standard approach for image classification tasks, particularly for handwritten digit recognition using the MNIST dataset [1]. However, implementing CNN operations at the assembly level presents significant challenges in terms of memory management, vectorization, and computational efficiency.Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

should be typed in either Times New Roman or Symbol font, This project addresses the challenge of implementing a complete CNN forward pass using RISC-V vector instructions, specifically targeting MNIST digit classification. The RISC-V instruction set architecture (ISA) provides vector extensions that enable parallel processing of data, making it suitable for the computationally intensive operations required in neural network inference [2].

The primary objectives of this project include: implementation of convolution operations using vectorized instructions; development of efficient ReLU activation functions; creation of fully connected layer computations with matrix-vector multiplications; integration of all components into a complete CNN forward pass; and demonstration of practical machine learning inference at the assembly level.

1) The project implements a CNN architecture that processes 8×8 input images through convolutional layers with 8 filters, followed by ReLU activation, and two fully connected layers for final classification. This implementation provides a foundation for understanding how neural network operations can be optimized at the hardware level.

A. CNN Architecture Design

The implemented CNN architecture follows a standard forward pass structure optimized for RISC-V vector processing capabilities. The network consists of the following layers:

Input Layer: The system processes 8×8 grayscale images represented as floating-point values stored in the InputVector data structure. Each pixel value is stored as a 32-bit IEEE 754 floating-point number to maintain precision during computations.

Convolutional Layer: Eight 3×3 convolution filters are applied to the input image, producing 8 feature maps of size 6×6. The convolution operations utilize RISC-V vector instructions to process multiple pixels simultaneously, significantly improving computational efficiency compared to scalar implementations.

Activation Layer: ReLU (Rectified Linear Unit) activation is applied element-wise to the convolution output, setting all negative values to zero while preserving positive values. This introduces non-linearity essential for the network's learning capability.

Fully Connected Layers: Two dense layers perform matrix-vector multiplications:

- FC1: Transforms the flattened 288-element feature vector (8×6×6) to 10 neurons
- FC2: Processes the 10-neuron intermediate representation to produce final classification scores

Output Layer: An argmax operation identifies the neuron with the highest activation value, corresponding to the predicted digit class (0-9).

B. Vector Instruction Utilization

The implementation leverages RISC-V vector extensions for parallel processing of neural network operations. Key vectorization strategies include:

Vector Length Configuration: The vsetvli instruction dynamically configures vector length based on data requirements, optimizing register utilization for different operation types.

Parallel Convolution: Convolution operations process 3×3 patches using vectorized multiplication (vfmul.vv) followed by reduction operations (vredsum.vs) to compute filter responses efficiently.

Vectorized Memory Operations: Vector load (vle32.v) and store operations enable efficient data movement between memory and vector registers, reducing instruction overhead.

Reduction Operations: Sum reductions using vredsum.vs efficiently compute dot products essential for convolution and fully connected layer operations.

C. Memory Management Strategy

The implementation employs a static memory allocation strategy optimized for predictable access patterns:

Data Segment Organization: All neural network parameters (weights, biases) and intermediate results are allocated in the .data segment with explicit space reservations:

- ConvOut: 1152 bytes for convolution outputs
- ReLUOut: 1152 bytes for ReLU activation results
- Z1, Z2: 40 bytes each for fully connected layer outputs

Address Calculation: Memory addresses are computed using explicit arithmetic operations, ensuring predictable access patterns that maximize cache efficiency and minimize memory latency.

Buffer Management: A dedicated patch_buffer provides temporary storage for 3×3 convolution patches, enabling efficient vectorized processing of image regions.

III. IMPLEMENTATION DETAILS

A. Convolution Layer Implementation

The convolution layer represents the most computationally intensive component of the CNN implementation. The operation processes 8×8 input images using 8 different 3×3 filters to generate 8 feature maps of size 6×6.

The convolution algorithm employs a patch-gathering approach where 3×3 image patches are extracted and processed against each filter. The implementation uses vectorized operations to maximize throughput. After patch gathering, vectorized multiplication and reduction operations compute filter responses efficiently using vle32.v for loading patches and filters, vfmul.vv for element-wise multiplication, and vredsum.vs for sum reduction.

B. ReLU Activation Implementation

The ReLU activation function implements the mathematical operation f(x) = max(0, x) for each element in the feature maps. The implementation uses floating-point maximum operations (fmax.s) to efficiently handle the conditional logic, avoiding conditional branching that could impact performance.

C. Fully Connected Layer Implementation

The fully connected layers perform matrix-vector multiplications between weight matrices and input vectors, followed by bias addition. The implementation uses vectorized dot products to compute neuron activations efficiently through vector load operations, element-wise multiplication (vfmul.vv), dot product computation via reduction (vredsum.vs), and bias addition (fadd.s).

D. Prediction Generation

The final prediction is generated using an argmax operation that identifies the output neuron with the maximum activation value. This corresponds to the most likely digit class according to the network's learned representation.

II. IV. VECTOR INSTRUCTION ANALYSIS

Table I presents the core RISC-V vector instructions utilized in the implementation and their specific purposes within the CNN forward pass.

ORE VECTOR INSTRUCTIONS UTILIZED

Instruction	Purpose	Utilization Context
vsetvli	Configure vector length	Dynamic vector register sizing
vle32.v	Vector load 32-bit elements	Loading patches, weights, values
vse32.v	Vector store 32-bit elements	Storing computation results
vfmul.vv	Vector floating-point multiply	Element-wise multiplication
vredsum.vs	Vector reduction sum	Computing dot products
vmv.v.x	Vector move from scalar	Initializing vector registers
vmv.x.s	Scalar move from vector	Extracting scalar results

A. Vector Length Optimization

The implementation uses vsetvli x1, x1, e32, m1 to configure optimal vector length for 32-bit floating-point operations with multiplier 1, balancing register usage and computational throughput.

B. Memory Access Patterns

Vector load and store operations are aligned to maximize memory bandwidth utilization, with sequential access patterns that leverage cache efficiency.

V. OUTPUT AND VERIFICATION

A. Computational Verification

The implementation includes comprehensive verification mechanisms to ensure correctness of neural network computations. Intermediate result validation stores key values (convolution outputs, ReLU activations, FC layer results) in accessible memory locations for debugging and verification purposes.

B. Integration Testing

The complete CNN forward pass has been tested using representative input data. Input processing handles 8×8 test images through all network layers to verify end-to-end functionality. Output generation successfully produces classification predictions through the argmax operation, with results stored in register x4 for external verification.

C. Expected Output Format

The system produces several categories of output: classification results (predicted digit class 0-9 available in register x4), intermediate activations (stored in respective memory locations), and performance metrics through instruction counting and execution timing analysis.

VI. CHALLENGES AND SOLUTIONS

A. Memory Management Challenges

Efficient organization of large data structures including weight matrices, input images, and intermediate activations within assembly language constraints was addressed through static memory allocation with explicit space reservations and computed address offsets.

B. Vector Instruction Complexity

Effective utilization of RISC-V vector extensions for neural network operations was solved through systematic vector instruction usage, including dynamic vector length configuration and careful register allocation strategies.

C. Floating-Point Precision

Maintaining numerical accuracy throughout the CNN forward pass was ensured through consistent use of IEEE 754 single-precision floating-point arithmetic with proper handling of special cases.

D. Algorithm Vectorization

Adapting traditional CNN algorithms for efficient vectorized implementation was achieved through redesign of convolution and fully connected layer algorithms to leverage vector instructions effectively.

VII. PERFORMANCE ANALYSIS AND IMPROVEMENTS

A. Current Performance Characteristics

The implementation demonstrates efficient utilization of RISC-V vector extensions for CNN operations through vectorization efficiency in convolution and fully connected operations, memory bandwidth optimization via sequential access patterns, and improved instruction density through vector instruction usage.

B. Potential Improvements

Further optimization opportunities exist in: enhanced vectorization by expanding to ReLU activation and argmax operations; memory hierarchy optimization through data prefetching and cache-aware access patterns; instruction scheduling for reduced execution stalls; and precision optimization through mixed-precision arithmetic investigation.

C. Scalability Considerations

The implementation can be extended for input size scaling (larger images), increased network depth (additional layers), and filter optimization (more convolution filters) using established implementation patterns.

This project successfully demonstrates the implementation of a complete CNN forward pass using RISC-V vector instructions for MNIST digit classification. The implementation covers all essential components of a modern CNN, including convolution operations, ReLU activation, fully connected layers, and prediction generation.

Key achievements include comprehensive implementation of all major CNN components using efficient vector instructions, effective utilization of RISC-V vector extensions showcasing hardware-accelerated neural network inference potential, computational efficiency through vectorized operations and optimized algorithm design, and educational value bridging high-level neural network frameworks and hardware execution.

The implementation serves as a foundation for further research into hardware-accelerated machine learning, embedded AI systems, and specialized neural network processors. Future work could explore dynamic memory allocation, mixed-precision arithmetic, and integration with specialized AI accelerators.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[2] RISC-V International, "The RISC-V Vector Extension," Version 1.0, 2021.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] RISC-V International, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA," Version 20191213, Dec. 2019.