POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

PROJECT REPORT

Low level optimization of a Convolutional Layer in ResNet-8 on RISC-V

BSc Computer Science Engineering

AUTHOR: Luca Medea

COURSE INSTRUCTOR: Prof. Cristina Silvano

PROJECT SUPERVISOR: Dott. Tommaso Spagnolo

ACADEMIC YEAR: 2024-2025

DURATION: 31/03/2025-24/08/2025

1. Introduction

In recent years, the open-standard **RISC-V instruction set architecture** has emerged as a compelling alternative to proprietary ISAs, thanks to its modularity, extensibility, and openness. Its flexibility makes it especially suitable for domain-specific architectures, where performance, energy efficiency, and resource constraints are critical.

Owing to the widespread and growing deployment of artificial intelligence, **deep neural networks** are increasingly deployed in environments with strict performance and energy constraints. In this context, **computational efficiency** has become a key technical priority: achieving faster inference, lower power consumption, and minimal latency is essential for enabling real-time AI on embedded and edge platforms. These improvements not only enhance scalability and responsiveness but also reduce hardware stress and energy costs. To meet these demands exploiting **architectural features** is crucial, allowing for optimizations beyond the reach of conventional high-level implementations. This project aims to **develop and optimize a convolutional layer in RISC-V assembly**, demonstrating its superior performance compared to an equivalent implementation compiled from an high-level language such as C. Moreover, it explores the feasibility of integrating the **Strassen algorithm** into a convolutional layer, with the goal of investigating its practical benefits in reducing computational complexity on resource-constrained platforms.
The complete source code developed for this project is publicly available on GitHub:

https://github.com/lucamedeaa/ResNet8-RISC-V

2. Innovation

The innovation of this work stems from the combination of **low-level RISC-V assembly optimizations** and the **direct implementation of Strassen's algorithm** within a convolutional neural network. Unlike approaches focused exclusively on compiler-driven or high-level methods, this study demonstrates that carefully

crafted instruction-level techniques—such as scheduling, branch elimination, and full loop unrolling—can significantly accelerate convolutional execution on resource-constrained hardware.
In parallel, Strassen's algorithm is explicitly integrated into convolutional layers, representing a novel contribution in this context and enabling a rigorous evaluation of the trade-off between multiplication reduction and the overhead of additional summations and memory operations.
Taken together, these results highlight the potential of combining algorithmic decomposition with hardware-aware optimization, establishing a foundation for future exploration of efficient deep learning execution on open architectures such as RISC-V.

3. Background on CNNs

Convolutional Neural Networks (CNNs) are a subclass of deep neural networks optimized for structured input data, particularly images. Their architecture, as shown in Figure 1, is characterized by convolutional layers that apply spatially local, weight-shared filters across input tensors to extract hierarchical feature representations. Compared to fully connected layers, this design significantly reduces parameter count while preserving spatial information.
CNNs typically interleave convolutional layers with non-linear activation functions, such as ReLU, and may include pooling and normalization layers to improve generalization and training stability. These components can be grouped into compound modules, such as residual blocks, which facilitate the training of deeper architectures.
Due to their high arithmetic intensity and regular dataflow, convolutional layers dominate inference cost and are thus the primary target for architectural and algorithmic optimization in CNN-based systems.

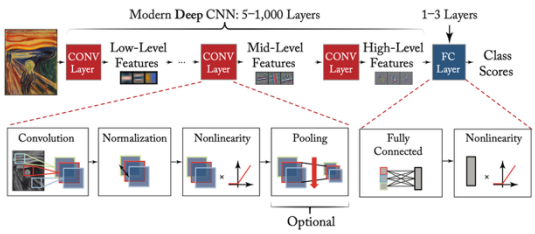

Figure 1: Structure of a deep CNN

4. Convolutional layer design

The convolutional layer implemented in this work takes as input a feature map of dimensions 32×32 with 3 channels. It applies 32 convolutional filters, each of size 3×3x3 corresponding to a 3×3 receptive field over each of the 3 input channels. The convolution is performed with unit stride and unit padding, resulting in an output tensor of size 32x32×3, computed according to Formula 1.

$$\text{output}[oc,h,w] = \text{ReLU}\left(\text{Quant}\left(bias[oc] + \sum_{ic=0}^{2}\sum_{kh=0}^{2}\sum_{kw=0}^{2} input[ic,h+kh,w+kw]\cdot kernel[oc,ic,kh,kw]\right)\right)$$

Formula 1: Convolution output

At each spatial position, a 3×3 region is extracted from each input channel and multiplied element-wise with the corresponding filter weights, as illustrated in Figure 2. The filter is then shifted across the input feature map with a stride of one, systematically covering all possible receptive fields. In this way, local spatial patterns are captured and transformed into feature activations. The 27 resulting products (9 spatial values × 3 channels) are summed to produce a single scalar. A bias term is then added, and the result forms one element of the output feature map. This operation is repeated across all spatial locations and for each of the 32 filters, generating the full output tensor.
In all implementations, variables are represented and processed using 8-bit signed integer (INT8) precision. This choice reduces memory footprint and bandwidth requirements while aligning with typical quantization strategies for efficient inference.

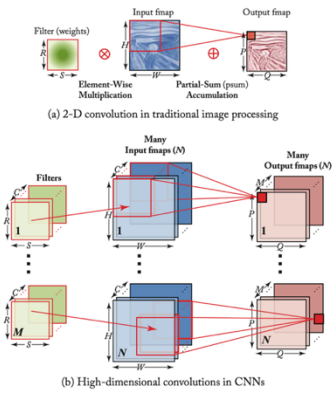

Figure 2: 2-D vs. CNN convolution

5. Low-level optimization

The development process began with the implementation of a reference **C version** of the convolutional layer. This step was not only essential for verifying functional correctness but also played a critical role in achieving a complete understanding of the algorithmic flow that would later be ported to RISC-V 64 assembly. The C implementation served as a baseline against which all subsequent performance measurements and correctness checks were conducted.

Following this, the **first assembly version (v1)** of Conv0 was implemented, designed as a straightforward and direct translation of the C logic. This "naïve" approach retained the full set of nested loops for the three input channels and the 3×3 kernel dimensions, performing bounds checking on each multiply–accumulate (MAC) operation. While functionally correct and structurally simple, this variant incurred a slight performance overhead due to repeated address computations and branch instructions within the innermost loops.

In the **second and final assembly version (v2)**, a set of targeted optimizations derived from both algorithmic analysis and architectural considerations of the RISC-V pipeline was systematically incorporated. Specifically:

1. **Full unrolling of the 27 MAC operations** per output pixel (3 input channels x 9 kernel elements), thereby eliminating loop control overhead and enabling better instruction-level parallelism.
2. Introduction of a **pre-computed halo buffer** of dimensions 34×34×3 surrounding the input tensor. This design completely removed conditional branch instructions from the MAC execution path by ensuring that all memory accesses during convolution were in-bounds.
3. **Pre-computation of base pointers for each input channel and weight block**, combined with stride-based pointer increments (addi), replaced repeated multiplication-based address calculations. This **strength-reduction** strategy significantly reduced the arithmetic workload inside the critical loop body.

These optimizations collectively resulted in a substantial reduction in cycle count relative to both the naïve assembly implementation and the C baseline, demonstrating the effectiveness of low-level control over data access patterns and loop execution in performance-critical convolutional layers, as per Figure 3.

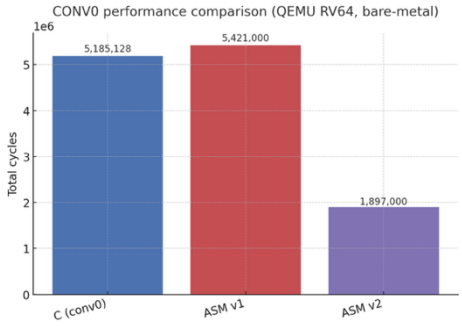

Figure 3: Performance Comparison

5.1. ResNet-8 Development

To ensure maximum comprehensiveness, a complete ResNet-8 in C has been implemented, achieving **2,25 millions of clock cycles**.
The architecture is composed of the following components:

- **Conv0 (Initial Convolutional Layer):** extracts low-level spatial features from the input image.
- **Residual Blocks (three groups):** each group contains two convolutional layers for feature transformation, bias addition for adjusting activation baselines, quantization for INT8 scaling, and ReLU activation for non-linearity.
- **Skip Connections:** identity mappings that bypass one or more convolutional layers to preserve information and improve gradient flow.
- **Average Pooling Layer:** reduces spatial resolution of the feature maps before classification, lowering computational cost.
- **Fully Connected Layer:** maps the final feature representation to the output classes.

All units were implemented in INT8 precision, ensuring consistency with the assembly-optimized convolutional layer and providing a reliable baseline for end-to-end performance assessment.

5.2. Strassen's algorithm

In the standard formulation, the product of two nxn matrices requires $O(n^3)$ scalar multiplications, following $c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$.
A divide-and-conquer approach splits each matrix into four $\frac{n}{2}\ x\ \frac{n}{2}$ blocks, yielding eight recursive multiplications. Strassen's algorithm reduces this number to seven by introducing linear combinations of submatrices. Specifically, after partitioning $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$,
ten auxiliary sums and differences are formed:

$$S_1 = B_{12} - B_{22}, \qquad S_2 = A_{11} + A_{12},$$
$$S_3 = A_{21} + A_{22}, \qquad S_4 = B_{21} - B_{11},$$
$$S_5 = A_{11} + A_{22}, \qquad S_6 = B_{11} + B_{22},$$
$$S_7 = A_{12} - A_{22}, \qquad S_8 = B_{21} + B_{22},$$
$$S_9 = A_{11} - A_{21}, \qquad S_{10} = B_{11} + B_{12}.$$

Using these, seven matrix products are computed:

$$P_1 = A_{11} \cdot S_1, \qquad P_2 = S_2 \cdot B_{22},$$
$$P_3 = S_3 \cdot B_{11}, \qquad P_4 = A_{22} \cdot S_4,$$
$$P_5 = S_5 \cdot S_6, \qquad P_6 = S_7 \cdot S_8,$$
$$P_7 = S_9 \cdot S_{10}.$$

Finally, the four quadrants of the product are reconstructed:

$$C_{11} = P_5 + P_4 - P_2 + P_6,$$
$$C_{12} = P_1 + P_2,$$
$$C_{21} = P_3 + P_4,$$
$$C_{22} = P_5 + P_1 - P_3 - P_7.$$

This process reduces the recurrence from: $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$, to : $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$,
Yielding an asymptotic complexity of:

$$T(n) = O\left(n^{\log_2 7}\right) \approx O(n^{2.81})$$

In the convolutional setting, each sliding patch–filter multiplication was restructured as a matrix product, enabling the direct application of this scheme. The implementation explicitly performs the block partitioning and auxiliary sums at the convolutional kernel level, thus embedding Strassen's decomposition into the core of the convolution operator.

5.3. Strassen Implementation

Motivated by an intuition, this work explored the potential of applying Strassen's algorithm for matrix multiplication within convolutional neural networks (CNNs), with the objective of improving execution time. To the best of our knowledge, no prior work had examined the integration of Strassen's method in this context on RISC-V platforms.

The first step was the implementation of the initial convolutional layer (*Conv0*) with Strassen applied at a single level of multiplication. This variant achieved **3.528.368 clock cycles**, outperforming the C baseline implementation. However, when extending Strassen to two levels of recursive multiplications, the results revealed a substantial overhead, amounting to $\mathbf{3,6 * 10^9}$ **clock cycles**.

It is important to emphasize that these variants did not include additional optimizations, as the experimental focus was solely on isolating the contribution of Strassen's algorithm itself. Finally, I integrated the one-level Strassen variant into the Conv0 of a complete ResNet-8 implementation. Interestingly, despite the performance improvement observed at the convolutional layer, the overall network exhibited no performance improvement, resulting in $\mathbf{2,28 * 10^6}$ **clock cycles,** which closely matches the ResNet-8 baseline.

These results underline the potential of Strassen-inspired hybrid approaches beyond the direct experiments conducted in this work.
Even though the application of Strassen across the full ResNet-8 resulted in a very slight performance degradation, the one-level Strassen variant of Conv0 still outperformed the C baseline, confirming that even small convolutional kernels can achieve measurable gains when multiplications are reduced.

This perspective is reinforced by the insightful research presented in *StrassenNets* (Tschannen et al., ETH Zürich, 2018), which showed that Strassen-like decompositions can be systematically integrated into deep learning models to reduce multiplications while retaining accuracy.Taken together, these insights suggest that Strassen-based methods, although limited in

the present setting, represent a compelling direction for future research and could play a decisive role in enabling efficient neural network inference on constrained hardware platforms.

6. Testing Environment

The testing and analysis environment is a RISC-V x64 system virtualized on QEMU. Clock cycle measurement is performed using the 'csrr' instruction: the current cycle count is stored in a register both before and after executing the convolution layer, and the difference is printed via UART. Since the code runs in a bare-metal environment, the output is obtained using a QEMU feature that allows inspection of register values at the end of execution.
RISC-V was chosen due to its growing importance as an open, extensible ISA with strong prospects in both research and industry. A prior in-depth study of the platform, combined with its relevance in resource-constrained environments, provided the ideal context for a low-level performance analysis.

7. Achieved results

The baseline C implementation of *Conv0* executed in **5,185,128 cycles**. Applying Strassen's algorithm at a single level reduced the cost to **3,528,368 cycles**, showing that a limited recursive decomposition can reduce the number of multiplications and improve performance. However, extending the decomposition to two levels required $\mathbf{3,7 * 10^9}$ **cycles**, where the overhead of additional sums and memory operations outweighed the reduction in multiplications.

The first assembly version (*v1*), obtained as a direct translation of the C code with nested loops and branch-based boundary checks, required **5.421.000 cycles**. The slight degradation compared to the C baseline can be attributed to frequent branch evaluations and repeated address arithmetic, which increased the instruction count on the critical path. The second and fully optimized assembly version (*v2*), achieved **1.897.000 cycles**. This represents a **2.73× improvement over the C baseline** and a **2.86× improvement over v1**, confirming the effectiveness of removing control-flow dependencies and redundant address calculations, as visualized in Figure 4.
At the network level, the baseline ResNet-8 executed in $\mathbf{2,25 * 10^8}$ **cycles**, whereas the Strassen-based implementation required $\mathbf{2,29 * 10^8}$ **cycles**. The near-equivalent performance indicates that multiplication savings are offset by summation and memory overhead, yet it also points to promising directions for new strategies that selectively exploit Strassen in combination with low-level optimizations.

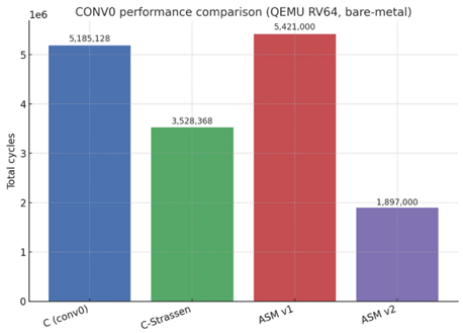

Figure 4: Conv0 testing outcome

7.1. Tables

Version	Clock Cycles	Speedup vs Baseline
C-Baseline	5 185 128	-
ASM v1	5 421 000	0.96x
ASM v2	1 897 000	2.73x
C-Strassen-1lev	3 528 368	1.47x
C-Strassen-2lev	3 654 760 280	0.0014x

Table 1: Conv0 Final Outcomes

Version	Clock Cycles	Speedup vs Baseline
C-Baseline	224 508 752	-
C-Strassen	228 988 000	0.98x

Table 2: ResNet-8 Final Outcomes

Note: Screenshots of the experimental outcomes are provided in a supplementary document.

8. Conclusions

The experimental results obtained throughout this project demonstrate that low-level optimization on RISC-V can deliver concrete and measurable improvements in the execution of convolutional layers. The optimized assembly implementation

reduced the execution time of the baseline convolution by more than a factor of two, a gain that is highly relevant in the context of embedded and resource-constrained environments. These results confirm that carefully designed instruction-level optimizations—such as eliminating branches, precomputing addressing logic, and unrolling inner loops—are not merely micro-optimizations, but effective strategies that significantly enhance overall efficiency. Importantly, the improvements were achieved without compromising correctness or precision, thereby underscoring the maturity and reliability of low-level design as a viable path to accelerate neural network inference.

The significance of these outcomes lies in showing that even when starting from a mature compiler baseline, deliberate low-level engineering can outperform automated code generation. This highlights the value of human-guided optimization in scenarios where performance and energy constraints are critical, and where each saved cycle translates into extended battery life or reduced latency. The findings therefore provide a strong basis for extending this line of research toward the systematic design of optimized kernels for broader classes of neural network operations on RISC-V and similar open architectures.

In parallel, the investigation of Strassen's algorithm emphasized the delicate balance between algorithmic decomposition and hardware-level efficiency. At the level of a single convolution, the one-level application produced clear improvements, whereas deeper recursive decompositions introduced substantial overhead, ultimately degrading performance at the full-network scale. Nevertheless, these results pave the way for hybrid strategies that combine classical convolution schemes with selective Strassen-based transformations.
A deeper analysis of how to apply this algorithm effectively, identifying the scenarios in which its integration can provide concrete performance benefits, remains an open research direction that could enable hybrid convolution strategies and unlock further efficiency gains.

It is also worth noting that, although to the best of our knowledge no prior work has applied Strassen's algorithm directly to CNN inference on RISC-V platforms, relevant studies have explored its potential in deep learning more broadly.
In particular, *StrassenNets: Deep Learning with a Multiplication Budget* by Tschannen et al. (ETH Zürich, 2018) demonstrated that Strassen-like decompositions can be learned end-to-end, reducing multiplications in ResNet-18 and NLP models by more than 99% while maintaining competitive accuracy. These results confirm that Strassen-type methods can be highly effective when applied at scale and in conjunction with learning-based strategies.

In this sense, the present work does not aim to provide a definitive judgment on Strassen's applicability, but rather to establish a first attempt at testing its integration in convolutional pipelines under strict resource constraints. The evidence collected here confirms both the limitations and the promise of this line of research: while Strassen is more suitable for large-scale networks, its combination with low-level techniques may ultimately prove fruitful. Future work in this direction will therefore be essential to fully exploit the potential of both architectural control and algorithmic innovation in advancing neural network efficiency on RISC-V.

9. Bibliography

[1] V. Sze, Yu-Hsin Chen, Tien-ju Yang, J. Emer "Efficient Processing of Deep Neural Networks".
[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, "Introduction to algorithms and data structures".
[3] C. Silvano, ACSO Course, Politecnico di Milano,2025.
[4] https://youtube.com/watch?v=aircAruvnKk

[5] https://youtube.com/watch?v=IHZwWFHWa-w

[6] https://youtube.com/watch?v=Ilg3gGewQ5U

[7] https://youtube.com/watch?v=tIeHLnjs5U8

[8] M.Tschannen, A.Khanna, A.Anandkumar, "StrassenNets: Deep learning with a multiplication budget"