Practical 1: Performance Benchmarking of 2D Convolution for Sobel Edge Detection

Samson Okuthe
Department of Electrical Engineering
University of Cape Town
GitHub Repository

Nyakallo Peete
Department of Electrical Engineering
University of Cape Town
GitHub Repository

***Abstract*—This report evaluates the computational performance of algorithmic implementations in image processing by comparing a manually coded 2D convolution against MATLAB's optimized built-in `conv2` function. Using Sobel operators for edge detection, execution times were benchmarked across five image resolutions ranging from 16,384 to 4,194,304 pixels. The results verified mathematical equivalence between both methods. Performance evaluation showed that while both implementations scale with the number of pixels processed for a fixed 3×3 kernel, the compiled and optimized `conv2` implementation achieved substantial speedup. Additional repeated-run experiments reveal that cold-start measurements can be heavily distorted by Just-In-Time (JIT) compilation, library initialization, and runtime overhead, especially for small matrices. Steady-state benchmarking demonstrates consistent acceleration for medium and large image sizes, with remaining variance attributable to system-level effects such as OS scheduling and dynamic CPU frequency scaling.**

***Index Terms*—Sobel edge detection, 2D convolution, MATLAB, performance benchmarking, image processing, cache effects**

I. Introduction

This practical explores computational performance by benchmarking a basic algorithmic implementation against an optimized library-based solution. Specifically, a manual 2D convolution implemented using strictly nested loops is compared to MATLAB's built-in `conv2` function for Sobel edge detection. Performance is evaluated across five image sizes to characterize how execution time scales with increasing input size, and to observe how system-level effects influence measured wall-clock time.

The Sobel operator estimates image gradients in two orthogonal directions, highlighting regions of high spatial frequency associated with edges. Implementing Sobel filtering requires repeated 2D convolution, which provides a clear demonstration of the cost of unoptimized scalar processing in an interpreted environment and the benefits of highly optimized numerical libraries.

II. Methodology

The project was developed in MATLAB to ensure a controlled environment for correctness verification and benchmarking.

A. Data Acquisition and Edge Handling

Five sample images of increasing resolution were pre-loaded into a cell array to decouple disk I/O latency from measured processing time. To ensure robustness, an edge-case handler was implemented to detect 3-channel RGB images and convert them to 1-channel grayscale. Images were cast to `double` precision to avoid arithmetic overflow during convolution and gradient operations. Sobel operators G_x and G_y were defined as 3×3 kernels to approximate horizontal and vertical gradient components.

B. Algorithmic Implementation

Two convolution methods were implemented:

- **Manual Implementation (`my_conv2`):** A custom function using strictly nested `for` loops. The kernel was rotated by 180° to implement true convolution (not correlation). Manual zero-padding was applied to preserve input dimensionality and produce a `'same'`-sized output.
- **Built-in Implementation (`inbuilt_conv2`):** A wrapper around MATLAB's `conv2` using the `'same'` option for consistent output shape, enabling a fair comparison with the manual method [1].

C. Benchmarking Strategy and Verification

Benchmarking was performed using `tic`/`toc` wall-clock timing. Two complementary timing viewpoints were used:

- **Cold-start (First-run) timing:** Captures initialization overhead such as JIT compilation, library loading, and runtime setup. These effects can dominate timing for small arrays.
- **Steady-state timing (Repeated runs):** Subsequent runs after initialization provide a clearer measurement of computational throughput. Remaining variance is influenced by OS scheduling noise, background activity, cache state, and dynamic frequency scaling. This aligns with benchmarking cautions discussed in course material [2].

Correctness was validated by comparing manual and built-in outputs using the maximum absolute pixel-wise difference threshold of $< 1 \times 10^{-6}$.

III. Results

All experiments passed verification across all image sizes, confirming that the manual convolution implementation is mathematically equivalent to MATLAB's built-in method under the selected boundary handling.

Table I presents representative steady-state results (3rd run), where most cold-start overhead has been removed. Table II presents the first-run results, which include initialization overhead and show strong distortion for the smallest image size. Table III presents the 5th run, illustrating that runtime variance persists even after warm-up.

TABLE I
BENCHMARKING RESULTS (3RD RUN: STEADY-STATE BEHAVIOR)

Image Size	Total Pixels	Manual (s)	Built-in (s)	Speedup
128×128	16,384	0.0024	0.0002	13.68x
256×256	65,536	0.0074	0.0003	23.54x
512×512	262,144	0.0294	0.0021	13.91x
1024×1024	1,048,576	0.1327	0.0100	13.25x
2048×2048	4,194,304	0.4638	0.0306	15.16x

TABLE II
BENCHMARKING RESULTS (1ST RUN: COLD START / INITIALIZATION INCLUDED)

Image Size	Total Pixels	Manual (s)	Built-in (s)	Speedup
128×128	16,384	0.0114	0.0116	0.98x
256×256	65,536	0.0247	0.0008	29.91x
512×512	262,144	0.0384	0.0024	15.96x
1024×1024	1,048,576	0.0919	0.0075	12.32x
2048×2048	4,194,304	0.3848	0.0317	12.15x

TABLE III
BENCHMARKING RESULTS (5TH RUN: RUNTIME VARIANCE OBSERVED)

Image Size	Total Pixels	Manual (s)	Built-in (s)	Speedup
128×128	16,384	0.0016	0.0002	6.77x
256×256	65,536	0.0062	0.0004	16.30x
512×512	262,144	0.0298	0.0019	15.95x
1024×1024	1,048,576	0.1042	0.0105	9.88x
2048×2048	4,194,304	0.4088	0.0290	14.12x

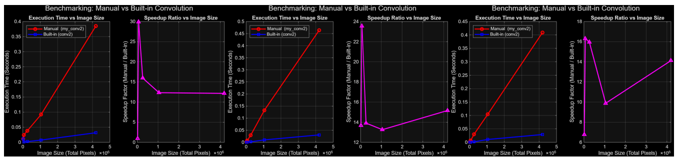

Fig. 1. Combined plot of execution time and speedup ratio from cold-start (1st run) through 5th run. Placeholder image: `Warmupto5th.png`.

IV. DISCUSSION

The benchmarking results demonstrate both algorithmic performance differences and system-level measurement effects.

A. Scaling Behavior and Constant Factors

For a fixed 3×3 Sobel kernel, both implementations scale proportionally with the number of pixels processed (i.e., linear in total pixels). However, the constant factors differ substantially. The manual method executes convolution using MATLAB-level loops, which incur interpreter overhead (loop control, indexing, and runtime checks). In contrast, `conv2` executes in optimized compiled code and is designed for efficient numerical throughput [1]. This difference explains the consistent order-of-magnitude speedups observed in the steady-state regime (Table I).

B. Cold-Start Distortion and Initialization Overhead

The first-run results (Table II) reveal a strong cold-start penalty, most clearly visible for the smallest image. For 128×128, the built-in function appears to provide little or no benefit (0.98x) because initialization overhead dominates the total measured time. Practical sources of this overhead include JIT compilation, library loading, runtime setup, and potential internal thread pool initialization. This confirms the importance of separating initialization effects from steady-state compute performance when benchmarking [2].

C. Steady-State Variance and System-Level Effects

While steady-state runs remove most initialization overhead, Table III shows that repeated runs can still vary, particularly for larger images. Such variance can be attributed to OS scheduling, background activity, cache state differences between runs, and dynamic CPU frequency scaling. These effects motivate statistical benchmarking approaches (e.g., repeated trials and median/mean aggregation) rather than relying on a single run. This aligns with broader perspectives on performance measurement and the need for accurate evaluation facilities in parallel computing systems [3].

D. Interpretation of Speedup Trends

Across medium and large image sizes, `conv2` consistently outperforms manual convolution by roughly one order of magnitude. Differences in speedup across sizes are expected: as images grow, memory traffic increases and the computation becomes increasingly constrained by data movement through the memory hierarchy. When memory bandwidth becomes a dominant factor, the relative advantage of arithmetic optimizations may reduce, even though the built-in method remains faster.

V. CONCLUSION

This practical successfully demonstrated the performance benefits of using optimized library functions for convolution-heavy image processing. Verification confirmed that the manual implementation was mathematically correct. Benchmarking results show that MATLAB's optimized `conv2` provides substantial speedup under steady-state conditions, while first-run timing can be misleading due to initialization overhead. The repeated-run analysis highlights that robust benchmarking should account for cold-start effects and runtime variance to provide representative performance estimates.

VI. AI CLAUSE

An AI language model was used to assist with interactive drafting and refinement of the report structure, benchmarking interpretation, and clarity of explanations regarding convolution scaling, steady-state measurement, and initialization overhead. All code and experimental results were implemented and validated by the authors on local hardware.

References

[1] MathWorks, "`conv2`: 2-D convolution," MATLAB Documentation. [Online]. Available: https://www.mathworks.com/help/matlab/ref/conv2.html. Accessed: Feb. 2026.

[2] S. Winberg, "Lecture 5: Performance Benchmarking & Wall Clock Timing," EEE4120F High Performance Embedded Systems, University of Cape Town, Lecture slides, 2026.

[3] S. Winberg, "Seminar 1: The Landscape of Parallel Computing (Research: A View from Berkeley)," EEE4120F High Performance Embedded Systems, University of Cape Town, Seminar slides, 2026.

Appendix A
Aggregated Median Benchmarking Results

To provide a compact and statistically representative summary under length constraints, an aggregated steady-state median table is included here. Cold-start measurements were excluded to avoid initialization bias. The medians below were computed using the warm runs available (3rd and 5th runs), which reduces sensitivity to outliers caused by OS scheduling and dynamic frequency scaling. The median speedup is computed as the ratio of the median manual time to the median built-in time.

TABLE IV
Aggregated Steady-State Median Results (Computed from Warm Runs)

Image Size	**Total Pixels**	**Manual Median (s)**	**Built-in Median (s)**	**Median Speedup**
128×128	16,384	0.0020	0.000 20	10.00x
256×256	65,536	0.0068	0.000 35	19.43x
512×512	262,144	0.0296	0.002 00	14.80x
1024×1024	1,048,576	0.1185	0.010 25	11.56x
2048×2048	4,194,304	0.4363	0.029 80	14.64x