I. J. Engineering and Manufacturing, **2026, 5, 227-237**
Published Online on October 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijem.2026.05.12

The Efficiency of FPGA-based Matrix Multiplication Accelerators for Neural Network Algorithms

Yaroslav Klyatchenko*
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 03056, Ukraine
E-mail: k_yaroslav@ukr.net
ORCID iD: https://orcid.org/0000-0003-4236-4059
*Corresponding Author

Oxana Tarasenko-Klyatchenko
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 03056, Ukraine
E-mail: o.tarasenko-kliatchenko@kpi.ua
ORCID iD: https://orcid.org/0000-0001-7200-0913

Received: May 7, 2026; Revised: June 15, 2026; Accepted: September 7, 2026; Published: October 8, 2026

Abstract: The contemporary challenges facing computer engineering have led to a focus on improving the efficiency of hardware and computing systems, as well as on optimising the operation of devices for effective data processing. In most modern systems of artificial intelligence, computer vision and digital signal processing, matrix multiplication is a basic operation. With the increasing resolution of sensors and the growing complexity of neural networks, classical general-purpose processors face the 'von Neumann bottleneck', where the data transfer rate between memory and the processor is limited, and the sequential execution of instructions does not allow the required real-time throughput to be achieved. The subject of this research is a System-on-a-Chip architecture that combines a dual-core processor based on the ARM architecture with programmable logic. This hybrid structure allows the most computationally intensive tasks to be offloaded to the hardware, whilst leaving control and the implementation of high-level interfaces to software. A neural network accelerator based on programmable logic devices offers advantages such as the capability for stream processing, which minimises the number of accesses to external memory, and the ability to utilise so-called mixed-precision computing. Furthermore, the accelerator's efficiency is achieved through the use of multiple processing elements, enabling parallel computation of the neural network's output channels. It has been demonstrated that implementing matrix operations on programmable logic enables parallelism of hundreds of operations per clock cycle, which is unattainable for systems based on general-purpose CPUs of the same class. The proposed implementations for a programmable logic-based accelerator have demonstrated speed-up compared to a system based on an ARM-architecture processor, as well as high adaptability to Deep Learning tasks.

Index Terms: Programmable Logic, System on a Chip, Matrix Multiplication, Convolutional Neural Networks, Denoising, Deep Learning

1. Introduction

At the current stage of scientific and technological development, computer image processing and intelligent image analysis are critical tools for many fields of activity and scientific research. Since classical computer vision algorithms often prove to be of limited effectiveness in the face of the high variability and complexity of real-world scenes, the introduction of neural network architectures has brought about a paradigm shift in approaches to image processing.

Convolutional Neural Networks (CNNs) [1] occupy a leading position in contemporary research. Their ability to automatically identify relevant features significantly exceeds the capabilities of methods based on manual feature determination. The multi-level hierarchical structure of CNNs ensures high accuracy in recognising complex patterns in visual data. Methods for enhancing image resolution (so-called super-resolution) and tools for automating the pre-processing of data sets, which optimise the preparatory stages of research, deserve special attention. Furthermore, neural network technologies are key in the field of digital forensics for verifying the authenticity of images and protecting against

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

manipulation. In such computer vision systems, Programmable Logic Devices (PLDs) function as high-performance computing units, enabling the conversion of unstructured video data into formalised features or real-time classification results. This enables the implementation of neural network models in autonomous embedded systems with strict latency requirements.

2. Problem statement and Literature review

Modern computer graphics relies on rendering techniques that take complex physical effects into account in order to achieve photorealism. However, their application is often limited by high computational costs. In particular, when using the Cycles renderer [3], the problem of residual noise arises. Eliminating this noise using traditional methods requires a significant increase in the number of samples, and consequently − additional time and hardware resources. As an effective solution to this problem, the authors of [2] proposed the use of a denoising autoencoder. The results of the experiment showed that such a model not only significantly reduces rendering time but also demonstrates the ability to transform image representations in accordance with the specifics of the training data.

The issue of the effectiveness of autoencoder architectures in the context of complex stream data configurations is examined in more detail in [4]. The authors researched the properties of neural networks using synthesised sets of two-dimensional images based on three-dimensional scenes comprising primitives (spheres, cubes, cylinders). An important aspect of this work was the analysis of the network's convolutional properties for object monitoring tasks, as well as an assessment of the correlation between the complexity of the input configurations and the model's training time.

At present, convolutional algorithms [5] are the most widely used in the field of digital signal processing due to the relative simplicity of their hardware implementation and the high efficiency of their filtering operations. In view of this, a priority task is to examine in detail various types of neural networks to identify the most suitable architectures capable of ensuring optimal convolutional image processing within the scope of the given tasks.

Convolution kernels form the basis of convolution-based image processing. The filtering of an m×n image using convolution can be represented by expression (1):

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{1}$$

where a, b non-negative integers (offset limits relative to the current pixel);

x, y coordinates on the image;

w the convolution kernel (m×n);

f the image [6].

At the same time, the main components of a CNN [7] are convolutional layers, dilation layers, reconstruction layers, and weighted convolutional kernels.

In a CNN, a convolutional kernel consists of weights that are randomly initialised at the start of the neural network's training and are adjusted during the training process. After passing through convolutional filters, the image transitions from one representation to another [8]. Figures 1–3 illustrate various data structures that require the use of corresponding convolutional kernels. Thus, convolutional kernels for one-dimensional structures will be one-dimensional, for two-dimensional structures — correspondingly two-dimensional, and so on.



Fig. 1. A convolution that generates the value of the output vector via the weighted sum of the input values: a) an example of forming one element of the output vector from the 'weighted' elements of the input vector (vector convolution), b) a convolution kernel (1x3)

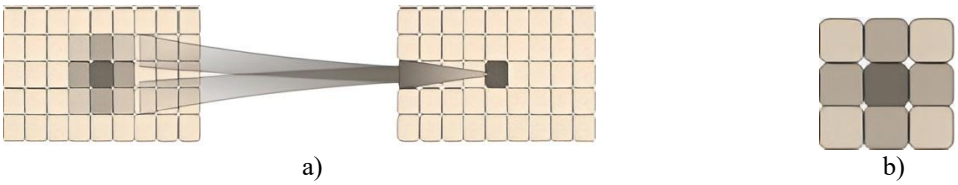


Fig. 2. Convolution that forms the value of the output two-dimensional array via the weighted sum of the input values: a) convolution of a two-dimensional array for a single element of the output array, b) convolution kernel (3x3)

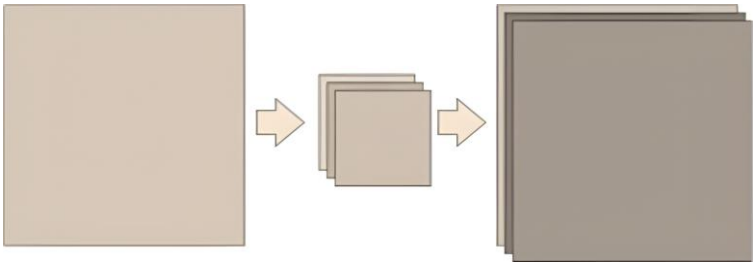


Fig. 3. Use of three convolution kernels

Deep learning neural networks, designed for deep learning tasks, utilise a large number of layers and neurons. During training, these neurons must adjust their weights based on the loss estimate at the output of the result layer. Such neural networks use a backpropagation algorithm, which employs gradient descent to adjust the weights of the neurons [9]. Convolutional neural networks use convolutional layers defined by:

-input maps (I);
-a set of filters (K);
-a set of basis values (b).

The output convolutional layers can be represented by the following expression (2):

$$S(i,j) = (I \times K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) + b \quad (2)$$

where i, j — integers;
m, n — the height and width of the filter [9].

New representation layers are formed by processing the previous layers through convolution filters. The number of new layers depends on the number of filters applied. For example, for spatial filtering of digital images using convolution filters, convolution kernels with typical sizes of 3×3, 5×5, 7×7, and so on are used. Generally, 3×3 and 5×5 kernels are used. However, for image restoration or non-standard resizing using convolution algorithms, not only the kernels mentioned above are used, but also kernels of sizes 2×2, 4×4, 8×8, etc. [10-12].

It is clear, therefore, that image and speech recognition tasks are a type of machine learning task, which is a form of deep learning, and require more powerful hardware for rapid processing. This hardware performs specialised operations to boost the data processing performance of artificial neural networks. For example, one such device is Google's specialised tensor processor, which incorporates a systolic array of matrix processor elements for performing matrix multiplication [13]. Matrix multiplication is implemented in such a tensor processor using a matrix of multiplication elements and the accumulation of computed values. The weight matrix and the activation matrix are input to this matrix, and the result is accumulated in accumulators. To multiply B input activation elements by an NxN weight matrix, 2×N+B processor cycles are required.

Some of the main hardware components used to execute artificial neural network algorithms include general-purpose processors, such as the Central Processing Unit (CPU), which contains several arithmetic logic units (ALUs), the Graphics Processing Unit (GPU), which has dozens of ALUs combined into vectors, and specialised units – the Tensor Processing Unit (TPU), which uses a systolic array for matrix multiplication [14].

For example, a GPU enables the implementation of systems based on artificial neural networks that can operate in soft real-time modes [15,16], which is achieved due to high-level parallelism and the presence of a large number of single-core ALUs.

As there is no direct implementation of the convolution operation in the specified hardware, there is a need to use special algorithms that allow matrix multiplication to be employed for implementing convolutional neural network algorithms. To this end, the input image matrix and the convolution kernel weight matrix are transformed. The results of this multiplication are the convolution results. One of the popular algorithms for such a transformation is the General Matrix Multiply (GEMM) operation [17].

In [18], a comparative analysis of neural network training speeds is presented using the example of an object recognition (classification) task. This task was performed on various hardware platforms. The popular MNIST image dataset was used as the training data. The analysis showed that processors designed to perform a specified number of floating-point operations per second consume a certain amount of energy. However, the figures obtained differ from those expected, and this may be due to a discrepancy between the hardware and software components (see Fig. 4 and Fig. 5).

The following are among the key technical specifications of hardware designed to execute artificial neural network algorithms:

- bit width, which ensures greater precision in floating-point operations;
- the ability to perform parallel processing;
- the presence of specialized hardware units for performing filtering operations;
- the ability to work with large amounts of memory.

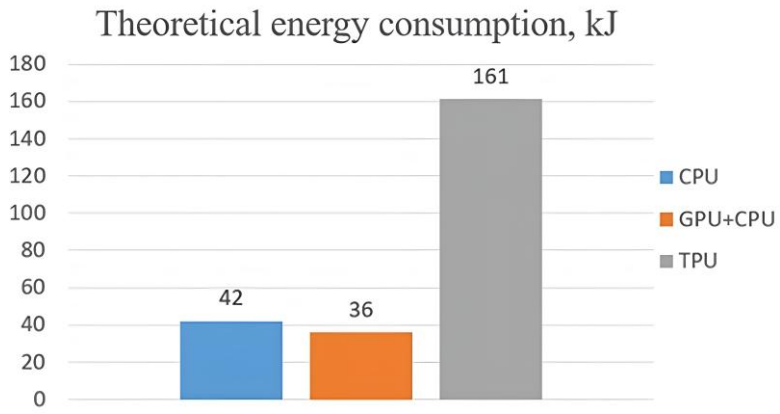


Fig. 4. Theoretical energy consumption by hardware during neural network training

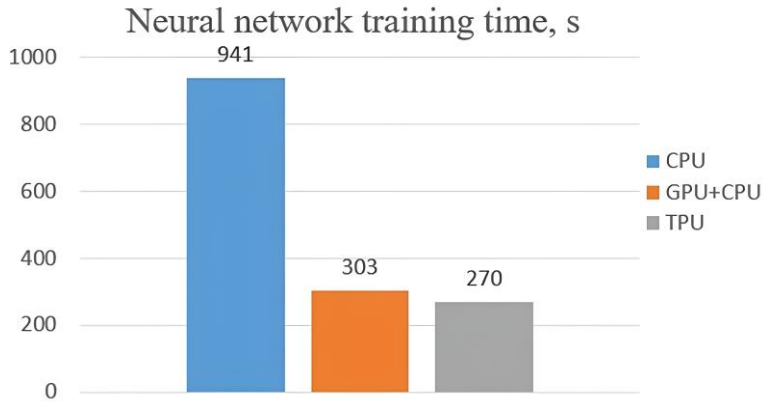


Fig. 5. Training time for a neural network using various hardware configurations

In general terms, hardware acceleration allows the execution of specialised tasks to be transferred from the CPU to specialised hardware components. Unlike sequential data processing in the CPU, accelerators utilise the principle of parallelism, which significantly increases the system's performance, energy efficiency and throughput.

The main types of accelerators include devices based on PLDs, which allow algorithms to be implemented directly on the chip. The current paradigm for their development is shifting from low-level hardware description languages (HDLs) to high-level synthesis (HLS). The use of C/C++ within HLS significantly speeds up the design of complex systems, although it imposes certain restrictions on the use of standard software functions (dynamic memory, recursion, etc.). Comparative characteristics are given in Table 1.

Table 1. Comparative characteristics of HDL and HLS

Characteristics	HDL (VHDL, Verilog)	HLS (C, C++, SystemC)
Level of abstraction	Low (Register-Transfer Level — RTL)	High (Algorithmic level)
Description objects	Flip-flops, logic gates, connections	Algorithm logic
Development speed	Low (long writing and debugging cycle)	High (rapid prototyping)
Access to resources	Full control over every logic element	Indirect (via compiler directives)
Code compactness	Complex (architecture-dependent)	High (code can be used in software)
Optimisation potential	Maximum performance and energy efficiency	Depends on the quality of synthesis tools

Therefore, the task of utilising specialised hardware based on PLDs as matrix multiplication accelerators is reasonable and relevant. The objective of this work is to investigate and demonstrate in practice the advantages of using a hardware accelerator based on the Zynq-7000 SoC [19] for performing matrix multiplication operations as a key component of modern neural networks. To achieve this objective, it is necessary to consider aspects such as the architectural features of PLD-based hardware, algorithmic optimisation, performance evaluation and adaptation for Deep Learning.

3. The efficiency of the matrix multiplication accelerator

Traditional CPUs execute instructions sequentially. Even when utilising multi-core processing and SIMD extensions, such as ARM NEON [20], they are constrained by the von Neumann architecture, where the constant exchange of data between memory and registers becomes a 'bottleneck' – 'The von Neumann bottleneck'. Matrix multiplication [17] is an operation with a computational complexity of $O(n^3)$. For matrices of size 1024×1024, over a billion Multiply-Accumulate (MAC) operations must be performed. Since the analysis of the efficiency of using PLD-based specialised processors for linear algebra tasks is one of the most pressing topics in modern computer engineering, the Zynq-7000 System on a Chip (SoC) platform [21] serves as an excellent example for researching this efficiency. It combines the flexibility of software (Processing Systems, PS) with the parallel computing power of hardware (Programmable Logic, PL) (Fig. 6.).

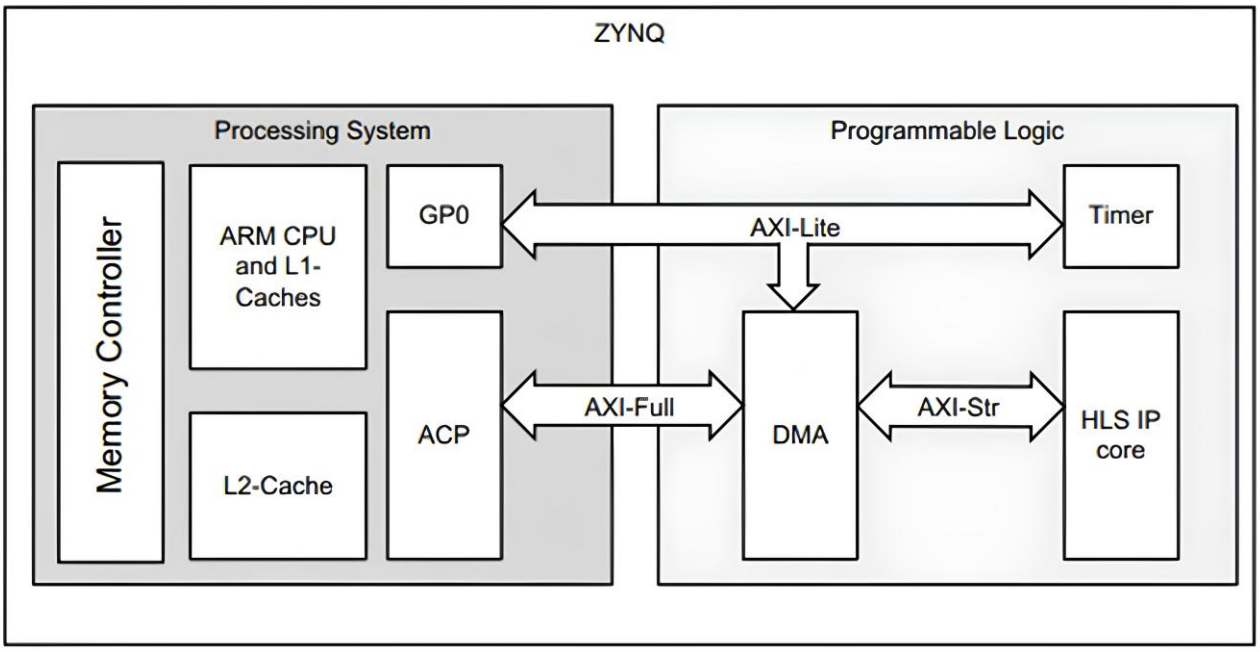


Fig. 6. PS and PL Partitions in the Zynq-7000 AP SoC [22]

As noted, the use of the Zynq-7000 SoC [19] represents a hybrid approach to implementing the accelerator. Specifically, a dual-core ARM Cortex-A9 manages the operating system, I/O and high-level logic. The programmable logic array forms the basis for a specialised parallel computation pipeline. The architecture of the matrix multiplication accelerator in the programmable section is based on a systolic array or parallel computation blocks using DSP48E1 [23]. Furthermore, the presence of internal ultra-fast Block RAM allows matrix fragments to be stored directly adjacent to the computational cores, minimising access delays to external DDR memory, and the high-speed bus (AXI4-Stream or AXI4-MM) used for data transfer between the ARM and FPGA [24] enables an increase in the efficiency of matrix multiplication for dimensions of 512x512 and larger.

For the general case, consider the features of software and hardware implementation using the example of multiplying two N×N matrices. A disadvantage of the software approach (for example, on an ARM Cortex-A9) is that even at a clock frequency of 667 MHz or 800 MHz, the processor spends many cycles fetching instructions, loading data from the L1/L2 cache or DDR, managing cycle counters and saving the result. As a result, the theoretical peak performance is only 10–15%. As for hardware implementation based on programmable logic, a specialised processor is created using an FPGA, which features parallelism (32, 64 or even 128 elements of the resultant matrix are computed simultaneously) and deterministic latency – each result appears exactly after a specific number of clock cycles.

The following (Fig. 7) is an example of C++ code for matrix multiplication with optimization directives (Vivado HLS) [25]. For efficient matrix multiplication C = A×B, the so-called tiling method (block multiplication) is used, as there are limitations on the capacity of the high-speed BRAM. A large matrix is divided into small 'tiles', which are placed in the FPGAs cache and processed sequentially [22,26-28].

To demonstrate the effectiveness of such an accelerator on programmable logic, it is proposed to consider the calculation of resource consumption using the example of the Zynq-7000 series SoC (XC7Z020) [21], which contains:

- DSP Slices: 220
- BRAM: 140 blocks (36 Kbits each)
- look-up tables (LUTs): 53,200
- Flip-Flops: 106,400.

```cpp
#include <ap_int.h>
#include <hls_stream.h>

#define N 32 // Matrix size for example

typedef float data_t;

void matrix_mul_accel(
    data_t A[N][N],
    data_t B[N][N],
    data_t C[N][N]
) {
    // Directives for creating interfaces (AXI4-Lite for control, Master for data)
    #pragma HLS INTERFACE s_axilite port=return bundle=CONTROL
    #pragma HLS INTERFACE m_axi port=A offset=slave bundle=DATA_A
    #pragma HLS INTERFACE m_axi port=B offset=slave bundle=DATA_B
    #pragma HLS INTERFACE m_axi port=C offset=slave bundle=DATA_C

    // Local buffers (BRAM)
    data_t local_A[N][N];
    data_t local_B[N][N];
    data_t local_C[N][N];

    // Splitting arrays for parallel access (Multi-port memory)
    #pragma HLS ARRAY_PARTITION variable=local_A dim=2 complete
    #pragma HLS ARRAY_PARTITION variable=local_B dim=1 complete

    // 1. Copying data from DDR to BRAM (Burst Read)
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            #pragma HLS PIPELINE
            local_A[i][j] = A[i][j];
            local_B[i][j] = B[i][j];
        }
    }

    // 2. Main computing core
    Row_Loop: for(int i = 0; i < N; i++) {
        Col_Loop: for(int j = 0; j < N; j++) {
            #pragma HLS PIPELINE // Inner loop pipeline
            data_t sum = 0;
            Product_Loop: for(int k = 0; k < N; k++) {
                // Thanks to ARRAY_PARTITION we access elements in 1 clock cycle
                sum += local_A[i][k] * local_B[k][j];
            }
            local_C[i][j] = sum;
        }
    }

    // 3. Copying the result back to DDR
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            #pragma HLS PIPELINE
            C[i][j] = local_C[i][j];
        }
    }
}
```

Fig. 7. Example of C++ code for matrix multiplication with optimization directives [22, 26]

The following are estimated calculations based on the information provided above. The determination of the number of DSPs is based on the assumption that, for a float multiplication operation, a full-featured MAC node occupies 3–5 DSP

blocks (depending on the Xilinx IP core). Therefore, if an internal cycle is unrolled, 32×3 = 96 DSP blocks will be required. This quantity is sufficient for an SoC that contains a total of 220 DSPs.

As for calculating the consumption of a high-speed resource such as BRAM, one must proceed from the following conditions: to multiply a 32×32 matrix with float-type elements (4 bytes), 32×32×4 = 4096 bytes (4 KB) will be required. However, a total of 3 such matrices (A, B and C) = 12 KB is required, which is equivalent to 3-6 blocks (one BRAM block 4 KB). This limitation is not critical when multiplying large matrices (for example, 1024×1024), as so-called tiling (processing in 32×32 fragments) is used.

The performance evaluation (by operations) for the case N=100 on an ARM Cortex-A9 CPU is as follows. As there are three embedded cycles here, 100^3 = 1,000,000 operations. Each operation (multiplication + addition + memory access) takes a minimum of 5–10 clock cycles (if there is no perfect cache hit). In total: ~10 million clock cycles.

Overall, for the FPGA (Programmable Logic) accelerator, we have the following results. Since the internal cycle produces one result per clock cycle, the calculation of 10 multiplications occurs in parallel. Therefore, the total number of clock cycles is: (100×100×100) / 10 = 100,000 clock cycles. As a conclusion, we can assume that at a frequency of 100 MHz (which is 6 times lower than the CPU), the FPGA accelerator will complete the task 10–15 times faster.

To ensure the practical relevance of the results obtained, it is necessary to perform calculations for problems of varying scales. This will enable a clear determination of the limits of FPGA efficiency compared to a CPU. When analysing performance, particular attention must be paid to the bandwidth limitations of the memory data exchange interfaces.

The following section examines the resource usage calculations for three typical matrix sizes: 128×128, 512×512 and 2048×2048. In doing so, we will assume that the same Zynq-7020 SoC, with the following specifications, is used for the multiplication of all the matrix dimensions listed (Table 2):

- Programmable logic (PL) frequency: 150 MHz.
- Number of DSP blocks: 200 units (parallelism P = 200 operations/cycle).
- Data type: FP16 (2 bytes).
- Theoretical peak performance: 150 MHz × 200 × 2 = 60 GFLOPS.

Table 2. Comparative table of matrix multiplication for FPGAs and CPUs (ARM)

Parameter	128x128 matrices	512x512 matrices	2048x2048 matrices
Number of operations	128^3=2.1 million	512^3=134.2 million	2048^3=8.59 billion
Computation time on FPGA	0.07 ms	4.5 ms	286 ms
Analysis of FPGA memory usage (BRAM)	Full (On-chip)	Partial + DMA exchange (Tiling)	Exchange with DDR required (swapping)
Computation time on CPU (ARM)	~1.5 ms	~90 ms	~6400 ms
Speedup	21x	20x	22x
Bottleneck	Computation frequency	Accelerator logic	Memory bus (DDR)

According to the data in the table, it can be concluded that the effectiveness of using FPGAs is not linear. This effectiveness is most evident in cases where the architecture allows memory latencies to be hidden behind the computational pipeline.

However, problems arise regarding data integrity and computational accuracy, stemming from the fact that when porting algorithms from general-purpose computers to FPGAs, it is necessary to choose a number representation format that affects both accuracy and memory usage. For example, standard neural networks are trained in the float32 format. However, implementing 32-bit floating-point arithmetic on FPGAs is resource-intensive (a single block for floating-point multiplication consumes a significant number of LUTs and DSPs) [8]. When converting to int8 (so-called quantization) the error can range from 0.5% to 2%, which is acceptable for many computer vision tasks but critical for scientific calculations. The FP16 (half precision) format was chosen as the working quantisation scheme, whereby operands fit entirely within a single DSP block to implement a full-fledged MAC unit. Consequently, this scheme has enabled the deployment of 200 independent parallel units without critical overloading of the look-up tables or the occurrence of delays due to routing congestion. This allows for maintaining a high dynamic range whilst using half the BRAM memory compared to float32. Using this reduced bit width allows, on the one hand, for an increase in the level of parallelism, because two int8 operations can be ‘packed’ into a single DSP block simultaneously (SIMD mode for DSP), and on the other hand, it is possible to reduce memory traffic. The hardware accelerator interacts with the DDR external memory subsystem via high-speed AXI HP ports. The m_axi interface, configured to operate in burst mode, is used for data array transfer ports to minimise the overhead associated with addressing. Control of the accelerator’s states and the transfer of scalar arguments, such as matrix dimensions, are carried out via the s_axilite interface. For example, transferring 8-bit

data instead of 32-bit data allows four times as many elements to be read in a single clock cycle via the same AXI HP port. However, there is a limitation in that, when multiplying large matrices, the accumulator must have a bit width higher than the input data.

4. Results and discussion

Despite the effectiveness of the proposed solution, the practical implementation of a Zynq-7000-based hardware accelerator faces a number of architectural limitations. The use of reduced-precision formats (FP16 or INT8) inevitably leads to quantisation errors. However, for most image classification tasks, the loss of accuracy is acceptable (within 1–2 %). This becomes a critical restriction for systems requiring high sensitivity. Such tasks include medical imaging or precision anomaly detection. Additionally, not all layers of neural networks cope equally well with bit-reduction, which may necessitate the use of a more complex mixed-precision architecture.

On the other hand, the system's bottleneck is the throughput of the external memory subsystem (DDR). The proposed tiling methods effectively mask delays only when there is a high degree of data reuse, which is characteristic of convolutional layers. However, when processing high-dimensional matrices in dense layers, this architecture is constrained by an I/O bound. In such cases, DSPs are idle whilst awaiting data, which negates the benefits of FPGA parallelism.

Taking these factors into account, the limits of applicability of such devices are clearly defined. The accelerator developed is suitable and highly efficient exclusively for the deployment stage of convolutional neural networks on peripheral devices (so-called Edge AI), which have power consumption constraints. However, its use is entirely unsuitable for the model training stage, which requires the float32 format. Furthermore, the accelerator cannot be used for deploying modern large language models, for example, whose requirements for memory bandwidth and parameter size significantly exceed the hardware capabilities of Zynq-7000.

5. Conclusion

Due to the increasing configurability of modern systems in many applications, such as those related to image processing, convolutional algorithms form the basis. Furthermore, the adaptability of computer (machine) vision systems can be achieved by using neural algorithms based specifically on multi-layer convolutional neural networks. Parallel computations for matrix multiplication provide effective hardware for image processing using artificial neural network algorithms. Since, for the typical 'average user', the most important requirements for the implementation of neural network-based systems are training speed and ability to process large volumes of data, it is important to focus on issues of parallel computing, bit width and the amount of memory used.

According to the results of the study into the performance of a hardware matrix multiplier based on the Zynq-7000 SoC architecture, the following conclusions can be drawn, covering the technical aspects of the development and the results of the calculations. The main conclusion of this work is the confirmation that a hybrid architecture, which can be described as 'processor + programmable logic', is optimal for embedded systems based on artificial intelligence. Traditional computing architectures (such as general-purpose CPU architectures) face a critical limitation: the sequential nature of instruction execution. Even when using vector extensions (such as the ARM NEON processor), the processor spends a significant portion of its power and time managing the instruction flow. In opposition to this, systems based on programmable logic allow for the implementation of an architecture where the data itself controls the computations. Real-time video stream processing requires not only high throughput but also minimal latency. This study demonstrates that transferring matrix operations to the programmable logic section allows for parallelism at the level of hundreds of operations per clock cycle, which is unattainable for mobile CPUs of a comparable class. Evaluation comparisons have been carried out, which have shown that a system based on the Zynq-7000 SoC offers the following advantages and features in computational processing:

- a system based on programmable logic (FPGA) allows pixels or matrix blocks to be processed in a stream, enabling the accelerator to begin producing the first matrix multiplication results even before the entire frame has been loaded into memory;
- real-time processing is achieved because connecting the CMOS sensors directly to the FPGA pins eliminates transmission delays over the system bus;
- optimisation of matrix multiplication (GEMM) involves the use of the Image to Column algorithm in combination with a mechanism for dividing large matrices into blocks that fit into local memory (BRAM), thereby transforming the convolution operation into highly efficient matrix multiplication at a rate exceeding 30 GFLOPS (using the XC7Z020 chip as an example).

Despite the high efficiency of multiplication operations based on a programmable logic system, a bottleneck has been identified – the bandwidth of the external DDR memory (due to its design characteristics). In this situation, even the fastest accelerator will be idle if the memory controller cannot provide the data in time. The solution to this problem lies

in maximising the use of internal BRAM blocks for caching neural network weights and the parallel use of multiple AXI HP ports. Experimental results from implementing the accelerator based on the Zynq-7000 SoC confirmed the possibility of achieving a 20-fold speed-up compared to a traditional software implementation based on a general-purpose CPU. Particular attention was paid to the configurability of the developed structure for neural network architectures of various types.

All the Declarations and Statements

Author Contributions Statement

Yaroslav Klyatchenko — Conceptualization, Methodology, Writing and Supervision: Conceptualization, Data curation, Reviewed and edited the manuscript, ensured clarity and coherence.
Oxana Tarasenko-Klyatchenko – Writing – Drafted the initial manuscript, contributed to the literature survey, and documented the technical background of the study.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare no conflicts of interest.

Funding Declaration

This research has not received any external financial support.

Data Availability Statement

This study analyzed publicly available datasets. The results obtained and the datasets are available to the author. The authors are ready to provide the dataset to anyone who needs it.

Ethical Declarations

Ethical approval was not required for this study as it did not involve human participants or animals.

Acknowledgments

We sincerely thank the reviewers for their professional evaluation and valuable recommendations, which have significantly contributed to improving the technical clarity and the overall quality of this paper.

Declaration of Generative AI in Scholarly Writing

During the preparation of this work, the authors used AI-assisted technologies to improve the linguistic clarity and formatting of the manuscript. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Abbreviations

The following abbreviations are used in this manuscript:

CNN - Convolutional Neural Networks
PLD - Programmable Logic Devices
CPU - Central Processing Unit
GPU - Graphics Processing Unit
FPGA - Field-Programmable Gate Array
GEMM - General Matrix Multiply

Appendix A/B/C…, with appendix title

None.

References

[1] Girish D. Chate, S. S. Bhamare, "Classification of Soil Images Using Convolutional Neural Network", International Journal of Image, Graphics and Signal Processing, Vol.17, No.5, pp. 26-41, 2025. DOI:10.5815/ijigsp.2025.05.03.

[2] L. Yasenko, Y. Klyatchenko and O. Tarasenko-Klyatchenko, "Image noise reduction by denoising autoencoder," 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Kyiv, Ukraine, 2020, pp. 351-355, doi: 10.1109/DESSERT50317.2020.9125027.

[3] Comparison Analysis of Cycles X Render and Cycles Render Using Google Colab. (2023). Jurnal Tika, 8(1), 90-94. https://doi.org/10.51179/tika.v8i1.1937.
[4] Yasenko, L., & Klyatchenko, Y. (2021). Convolutional properties of a neural network based on autoencoders. Information Technologies and Computer Engineering, 18(3), 77-85. https://doi.org/10.31649/1999-9941-2021-52-3-77-85.
[5] Ghimire, D., Kil, D., & Kim, S.-h. (2022). A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration. Electronics, 11(6), 945. https://doi.org/10.3390/electronics11060945.
[6] Gonzalez R.C., Woods R.E. Digital image processing. 2nd ed. Upper Saddle River, N.J: Prentice Hall, 2002. 793 p.
[7] Maladkar, K.: 6 Types of Artificial Neural Networks Currently Being Used in Machine Learning. Analytic Indian Magazine, 15 Jan 2018. https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/. Accessed 9 Oct 2020.
[8] P. Bharati and A. Pramanik, "Deep learning techniques—R-CNN to mask R-CNN: a survey," in Proc. Int. Conf. Comput. Intell. Pattern Recognit. (CIPR), 2019, pp. 657–668.
[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[10] Kondru Mounika, P. Venkateswara Rao, Anand Anbalagan, "Modified CNN Model for Network Intrusion Detection and Classification System Using Local Outlier Factor-based Recursive Feature Elimination", International Journal of Computer Network and Information Security, Vol.17, No.1, pp.82-91, 2025. DOI:10.5815/ijcnis.2025.01.07
[11] Devesh Kumar Srivastava, Chirag Goel, K. Kishore Kumar, Akhilesh Kumar Sharma, Babu R. Dawadi, Eshaan Saha, "Enhancing Underwater Object Detection through CNN-based Image Enhancement and Classification", International Journal of Engineering and Manufacturing (IJEM), Vol.16, No.2, pp.91-110, 2026. DOI:10.5815/ijem.2026.02.06
[12] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, “Review of image classification algorithms based on convolutional neural networks,” Remote Sensing, vol. 13, no. 22, p. 4712, 2021.
[13] Yilin Zhao Systematic Analysis on TPU’s History and Key Technologies Behind It // Proceedings of the 2025 International Conference on Electronics, Electrical and Grid Technology (ICEEGT2025), Advances in Engineering Research 292, https://doi.org/10.2991/978-94-6463-986-5_46/
[14] Takano S. Thinking Machines. Machine Learning and Its Hardware Implementation. Academic Press, 2021. 306 p.
[15] N. Skilkov and Y. Klyatchenko, "Complex methodology for determining WCET in modeling real-time system operation processes," 2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2024, pp. 1-6, doi: 10.1109/DESSERT65323.2024.11122223.
[16] Chellappa, R., Theodoridis, S. Academic Press Library in Signal Processing, Volume 7: Array, Radar and Communications Engineering. — Elsevier, 2021.
[17] Matrix Multiplication Background User Guide: NVIDIA Deep Learning Performance Documentation. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html.
[18] Klyatchenko Ya.M., Yasenko L.S. COMPARATIVE ANALYSIS OF THE EXECUTION SPEED OF NEURAL NETWORK ALGORITHMS DEPENDING ON HARDWARE. // APPLIED MATHEMATICS AND COMPUTING XIV Conference of Young Scientists PMK-2021. Kyiv: "PROSVITA", 2021. https://ela.kpi.ua/handle/123456789/55884.
[19] Zynq 7000 SoC Technical Reference Manual (UG585). https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM.
[20] Yang Zhang Arm Neon programming quick reference. Arm NEON programming quick reference guide. https://developer.arm.com/community/arm-community-blogs/b/operating-systems-blog/posts/arm-neon-programming-quick-reference.
[21] Zynq-7000 SoC Product Selection Guide (XMP097) URL: https://docs.amd.com/v/u/en-US/zynq-7000-product-selection-guide.
[22] A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS (XAPP1170). https://docs.amd.com/v/u/en-US/xapp1170-zynq-hls.
[23] AMD Adaptive Computing. 7 Series FPGAs DSP48E1 Slice User Guide (UG479). San Jose, CA: AMD, 2022.
[24] Xilinx Inc. Zynq-7000 SoC Technical Reference Manual (UG585). San Jose, CA: Xilinx, 2023. 1850 p.
[25] AMD Vivado™ High-Level Design URL: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/high-level-design.html#resources.
[26] Xilinx Inc. Vitis High-Level Synthesis User Guide (UG1399). San Jose, CA: AMD, 2024. 320 p.
[27] C. Jiang, D. Ojika, B. Patel, and H. Lam, “Optimized fpga-based deep learning accelerator for sparse cnn using high bandwidth memory,” in Proceedings of FCCM, 2021, pp. 157–164.
[28] Zhang, C., Li, P., et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. — Proceedings of the 2015 ACM/SIGDA International Symposium on FPGAs, pp. 161–170.

Authors’ Profiles



Yaroslav Klyatchenko Ph.D., associate professor of Department of System Programming and Specialized Computer Systems of National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine. Scientific interests: computer engineering, specialized computation, computer networks, machine learning.



Oxana Tarasenko-Klyatchenko PhD, associate professor of Department of System Programming and Specialized Computer Systems of National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine. Scientific interests: programming, computer logic, IoT, big data analytics, machine learning.

How to cite this paper

Yaroslav Klyatchenko, Oxana Tarasenko-Klyatchenko, "The Efficiency of FPGA-based Matrix Multiplication Accelerators for Neural Network Algorithms", International Journal of Engineering and Manufacturing(IJEM), Vol.16, No.5, pp. 227-237, 2026. DOI:10.5815/ijem.2026.05.12