Journal Pre-proof

Dynamic FPGA reconfiguration for scalable embedded artificial intelligence (AI): A co-design methodology for CNN acceleration

Jalil Boudjadar, Saif Ul Islam, Rajkumar Buyya

PII: S0167-739X(25)00072-X
DOI: https://doi.org/10.1016/j.future.2025.107777
Reference: FUTURE 107777

To appear in: *Future Generation Computer Systems*

Received date : 18 September 2024
Revised date : 8 January 2025
Accepted date : 18 February 2025

Please cite this article as: J. Boudjadar, S.U. Islam and R. Buyya, Dynamic FPGA reconfiguration for scalable embedded artificial intelligence (AI): A co-design methodology for CNN acceleration, *Future Generation Computer Systems* (2025), doi: https://doi.org/10.1016/j.future.2025.107777.

Dynamic FPGA Reconfiguration for Scalable Embedded Artificial Intelligence (AI): A Co-Design Methodology for CNN Acceleration

Jalil Boudjadar[a], Saif Ul Islam[b], Rajkumar Buyya[c]

[a]*Department of Electrical and Computer Engineering - Software Engineering & Computing systems, Aarhus University, Aarhus, 8200, Denmark*
[b]*WMG, The University of Warwick, Coventry, CV4 7AL, UK*
[c]*Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3125, Australia*

Abstract

In recent years, FPGA platforms have shown significant potential for accelerating artificial intelligence (AI) applications, particularly in Embedded AI. While various studies have explored adaptive AI deployment on FPGAs, there remains a gap in methodologies fully integrating software adaptability with FPGA hardware reconfigurability. This article presents a novel end-to-end co-design methodology for deploying adaptable and scalable Convolutional Neural Networks (CNNs) on FPGA platforms. The framework enhances computational performance and reduces latency by dynamically modifying hardware acceleration units by combining CNN architecture adaptability with dynamic partial reconfiguration of FPGA hardware. The proposed methodology enables automated synthesis and runtime customization of both hardware accelerators and CNN architectures, eliminating the need for iterative synthesis. This approach has been implemented and tested on a Xilinx XC7020 FPGA board for a CNN-based image classifier, achieving superior computation performance (0.68s/image) and accuracy (97%) compared to state-of-the-art alternatives.

Keywords: Adaptive CNNs, FPGA dynamic reconfiguration, Hardware acceleration, Co-design framework, Embedded AI, Computation performance, Scalable AI deployment.

1. Introduction

Over the last decade, Convolutional Neural Networks (CNN) [1] are used in solving complex problems such as classification, recognition, regression, prediction, and optimization [2, 3, 4, 5, 6, 7]. Solving complex tasks by mimicking the human brain and its biological neural network has been a topic for decades and was first opened up for debate in the 40s [8]. It is, however, only in the most recent years that CNN has been seen as a viable technology due to constraints enforced by the network sizes and the underlying computation complexity. This is due to the sheer amount of resources needed for utilizing CNNs [9] to their full potential, especially for computationally intensive models that utilize kernel filters to extract spatial information from images [10].

CNNs are deep learning models formed by several layers of neurons that rely on accumulating the knowledge mathematically from baseline training to infer decisions based on the input data [11]. Neurons of a layer are connected to some or all of the neurons from the adjacent layers to pass processing results. The neuron connections are weighted with coefficients to determine how much each input will contribute to the output in the next layer. Each neuron is associated with a bias value to be added to the output computation of the neuron. The weights and biases are computed through a training process [12].

CNNs extract features from the input images by recognizing the key patterns present in each image so as to classify them following the training and calibration of the CNN parameters so that the classification converges towards the pattern having the optimal output value [13]. CNNs may involve different image processing features such as segmentation, Max-pooling, and convolution. In fact, image segmentation enables an image to be processed in multiple smaller segments independently, where the calculations can be sent to dedicated hardware accelerators, a cluster of computational nodes, or a distributed system, thus achieving a short computation latency [14]. Max Pooling reduces a matrix of weighted pixels spatially into a smaller matrix by maintaining the highly informative pixels to reduce the computation cost without degrading the accuracy too much [15]. Convolution is the most computationally expensive operation as it amounts to multiplying an input image matrix with a kernel to generate a new image, called *fmap* [16]. The convolution works by sliding the kernel image over the input image, where each position of the kernel will generate a new pixel by multiplying each value in the kernel with the respective position of the kernel value in the input image. Convolution usually represents a fertile source for acceleration, given the many computation operations that can be performed in parallel.

Machine learning solutions, specifically CNN-based applications, often utilize cloud technology, where the actual network is deployed on powerful computation servers, making the inference speed relatively quick. For many application domains such as IoT and control systems, the deployment technology for CNN-based software solutions is recently undergoing a significant transformation, shifting from a cloud to edge and embedded computing [17, 18, 19, 20]. This was to enable 1) ubiquitous and pervasive computing and reduce the connectivity de-

pendency; 2) parallel execution of multiple layers and modules of the CNNs, thus reducing the execution latency and synchronizing with high-frequency data sources; 3) make it possible to process and extra features directly from the data source so that to reduce data communication cost and security threats; 4) coping with the privacy concerns and nascent GDPR.

However, an intrinsic challenge for the platform-aware design and adaptation of applications targeting FPGAs is the expensive synthesis cost for acceleration hardware and the lack of automated customization of the synthesized computation resources and CNN architecture at runtime [21]. For conventional acceleration frameworks, a new synthesis of the hardware acceleration cores is needed every time the deployed CNN changes because the acceleration cores are synthesized for a given software functionality [20, 22]. Another factor that can harden this challenge comes from the fact that a CNN can adapt its architecture at runtime (changes to the overall CNN structure, number of layers, size of layers, and other hyper-parameters) following changes in the input data space so that to reduce the computation burden and adjust the CNN functionality [23].

Over the last few years, a substantial effort has been devoted to accelerating CNNs on reconfigurable embedded platforms to process images [19, 24, 25, 26, 27, 21, 28]. However, to the best of the author's knowledge, none of the research studies considered optimizing the costly hardware synthesis operation and acceleration performance by *combining both CNN adaptability and FPGA reconfigurability in a single design approach* where both software and hardware are reconfigurable at runtime [29, 30], so that to leverage the execution performance and flexibility and reduce further the deployment cost of CNNs on FPGAs.

This article proposes a new co-design and deployment approach to leverage computation performance and latency for adaptive CNNs on FPGA platforms. The proposed approach enables iteration-free deployment to reduce the expensive cost of hardware accelerator synthesis. The accelerators are synthesized once and configured at runtime, using a combination of fine-grained and coarse-grained customization, following the adaptive CNN architecture. The CNN adaptability is secured upon an on-the-fly upload of new configurations (network structure and hyper-parameters) to the FPGA at runtime. The proposed design and deployment have been implemented to accelerate and deploy a CNN-based image classifier on a Xilinx ZYBO XC7020 FPGA. Computation performance, accuracy, resource utilization, and scalability are analyzed and compared to the state of the art. The major contributions of the article are summarized as follows:

1. A novel co-design and deployment framework that integrates adaptive Convolutional Neural Networks (CNNs) with FPGA platforms. The model eliminates the need for iterative synthesis, significantly reducing the costs and time associated with hardware accelerator deployment.

2. The proposed methodology leverages dynamic partial reconfiguration of FPGA hardware to accommodate changes in CNN architecture and hyper-parameters during runtime. This approach ensures the high adaptability of CNNs, enabling on-the-fly updates to the network structure and parameters.

3. By combining fine-grained and coarse-grained hardware customizations with adaptive CNN architecture, the model enhances computation performance and reduces latency. The co-design approach ensures that FPGA platforms can maximize the efficiency of hardware accelerators tailored to specific application requirements.

4. The proposed framework has been implemented and tested using a CNN-based image classifier on the Xilinx ZYBO XC7020 FPGA board. Detailed analysis of computation performance, accuracy, resource utilization, and scalability has been conducted, with results compared to state-of-the-art approaches to demonstrate the effectiveness and improvements of the proposed model.

5. The article contributes to the field of Embedded AI by presenting a comprehensive end-to-end methodology that combines CNN architecture adaptability with FPGA dynamic reconfiguration. This advancement facilitates deploying scalable and adaptable AI applications on FPGA platforms, pushing the boundaries of what is achievable in hardware-accelerated AI.

The rest of the article is organized as follows: Section 2 presents the state-of-the-art for acceleration architectural models and describes the relevant work. Section 3 explains the proposed methodology for customized acceleration of adaptive CNNs. Section 4 elaborates on the implementation and experiments using Xilinx XC7020 FPGA and compares the results to the state-of-the-art. Finally, Section 5 concludes the paper.

2. Related Work

A Field Programmable Gate Arrays (FPGA) is a computation platform composed of a conventional processing system (PS) and programmable logic (PL) [31]. PS is a conventional computer system that possesses processing cores (ARM processor), memories and caches. In contrast, PL is a set of fabric circuits (Flip-flops, registers, look-up tables, DSP, RAM blocks, etc.) that can be compiled to synthesize extra (user-defined) processing and storage components dedicated to executing given software functions. Compiling a set of circuits to implement the functionality of a software code as a hardware core is called High-level synthesis (HLS) [32]. The hardware components resulting from the HLS of a software function are called *Intellectual Property*, IP for short. IP cores are usually described using an HDL language such as VHDL or Verilog. They can be seen as functional blocks coupled with PS to execute a software system much faster by parallelizing and splitting the execution between PS and PL.

Machine learning-empowered systems are often deployed in dynamic environments. Being able to change the machine learning model post-deployment could be of capital interest to ensure high accuracy and performance through adaptability. CNN adaptation is an update of the structure, connections, weights,

and other hyper-parameters of the CNN architecture [24, 23, 6]. The adaptation can be performed offline or at runtime, mainly triggered by changes in the data stream, new training results, etc.

An IP core is synthesized for the network model to accelerate a CNN. Thus, runtime adaptability of a CNN may require re-synthesizing IPs and re-programing the FPGA, which is a complex and expensive task [25, 24]. A hardware IP reconfiguration is a customization of the generic IP functionality to execute a modified version of the original software used for the synthesis [21, 26]. Such automated customization is captured by tuning some of the IP parameters based on the benchmark attributes of the input software.

Strong effort has been devoted to implementing CNNs adaptability [26, 23, 20, 33] and runtime reconfiguration of FPGA hardware accelerators [24, 28, 21, 20, 34]. However, the literature still lacks adequate tooling and studies that tie the CNN adaptability and the hardware reconfigurability to design and deploy customizable CNN hardware accelerators [35, 24, 29].

Bouazzaoui *et al* [26] proposed a partial (coarse-grained) reconfiguration environment to accelerate the execution of machine learning models on FPGAs using dynamic classifier selection. Each classification model is implemented as a static accelerator to be activated and parsed to specific inputs at runtime. The identification of the most fitting classifier for incoming data, based on the K-Nearest Centroid approach, at runtime has led to considerable reduction in the resources utilization of the FPGA platform. Huang *et al* [36], the authors introduced a partial (fine-grained) reconfiguration architecture to accelerate CNN on FPGAs. The proposed acceleration relies on reconfiguring specific convolution layer hardware blocks according to the input model parameters at runtime. The partial reconfiguration of the IP blocks has led to high computation efficiency and decreased energy consumption.

Tong *et al* [37] presented a highly unified generic acceleration architecture to accelerate different machine learning models such as standard CNNs, lightweight CNNs and CNNs with DeCONV layers by dynamically reconfiguring the hardware accelerator following the architecture parameters of the input model. The acceleration architecture model reduces the overhead when deploying different models and enhances the overall resources utilization efficiency.

Kumar *et al* [20] proposed a hardware customization architecture to execute CNN layers and enable intelligent edge computing. To cope with the data transfer bottleneck, the layers execution is performed using a linear task model. However, executing layers as a sequence may lead to higher latency and degrade the computation performance, notably, if the hardware accelerators enable overlapped execution [29].

To leverage FPGA flexibility for CNN applications, a dynamic model enabling runtime reconfiguration of FPGA hardware to accelerate different CNN architectures was proposed in [22]. Using a layer-clustering algorithm, the authors classify the CNN layers and generate optimal hardware configurations to execute each layer. However, one must run the expensive layers classification for each update to the CNN architecture.

Wang *et al* [38] developed a scalable and cost-efficient FPGA accelerator for large-scale deep learning networks through a pipeline of three processing units to scale the performance and improve the throughput. However, data communication between the processing system and the acceleration units (PL) is a bottleneck [39]. To loosen the communication bottleneck between the FPGA processing system and the acceleration hardware, Shi *et al* [28] proposed an acceleration model with optimized dynamic allocation, through a classification of layers, to hardware processing elements using AXI bus interfaces. However, the classification processing overhead contributes to degrading the execution performance.

Ratto et al [35] proposed a toolchain to enable model-based adaptivity of CNNs and runtime reconfigurabilty of the underlying hardware accelerators. The proposed deployment relies on fine-grained reconfiguration of the hardware accelerators synthesized using ONNX parser and Vivado HLS by activating the subset of IP circuits corresponding to the functionality parameters in the CNN customization.

Zaidy *et al* [40] developed an efficient, low-power accelerator to leverage the inherent parallelism in CNN architectures. The computation efficiency resulted from implementing a set of *ComputeCore* accelerators each of which integrates the maps, weights buffers and comparators. One can see that, the computation efficiency is achieved on the expense of hardware size area which could be a bottleneck for scalability. Meloni *et al* [41] proposed a flexible hardware/software solution to accelerate CNNs on Zynq SoCs via an efficient allocation of the Zynq ARM cores to hard-to-accelerate tasks whereas CNN computations are allocated to the hardware accelerator. Although the accelerator is static and the CNN architecture does not change at runtime, a flexibility results from the dynamic scheduling of the computation resources and control of the accelerator.

In this article, we develop an agile deployment model for accelerating CNNs using FPGA hardware. The proposed method involves altering the functionality of FPGA by partially reconfiguring its hardware resources at runtime using both fine-grained (partial reconfiguration of IP cores) [36] and coarse grained customization (dynamic mapping of IP cores) [26, 42]. By combining CNN adfaptability and FPGA reconfigurability, our proposal enables users to easily shape adaptivity at model level, achieving thus application-specific HW accelerators. Specifically, we advance the hardware reconfiguration models in [36, 26, 22] by enabling the CNN to change its architecture at runtime, for which the hardware accelerators are *automatically* customized and *dynamically* mapped to secure high computation performance. Compared to the state of the art, the proposed co-design achieved a highly resource- and computation-efficient classification (0.68 second per image) while delivering one of the highest accuracy levels (97%).

3. Adaptive Acceleration Methodology

This section specifies the adaptive CNN architecture model and elaborates on how the hardware accelerators are customized at runtime following changes in the CNN template to leverage the execution performance.

The proposed methodology is depicted in Fig. 1. The hardware IP cores for acceleration are synthesized once at the design stage, from the software functionality using HLS, as standalone units flexible enough to accommodate runtime changes and integration of different applications. The CNN parameters are stored in on-chip memory upon adaptation to leverage performance and achieve less memory access [43]. The red components in the figure are dynamic. The blue arrows are reconfiguration to the IP cores, and the dashed arrows are dynamic allocations of the computation tasks to the IP cores.

We also developed an optimized dynamic allocation of CNN layers to the accelerators to leverage the performance and latency [42], enabling thus a coarse-grained reconfiguration. Besides, a fine-grained reconfiguration is achieved upon specifying different activation functions within each IP core, where the corresponding functionality is activated using the CNN layer parameters [35]. Although this may lead to a large hardware area for each accelerator, it enables low-cost customization and high flexibility.

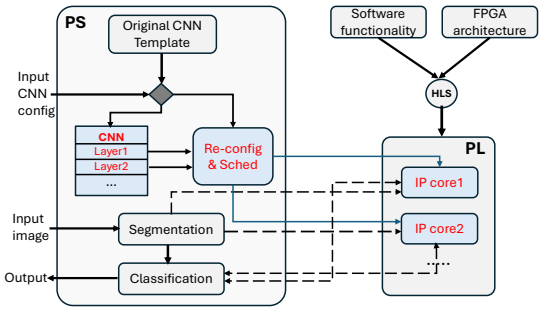


Figure 1: Proposed acceleration and deployment methodology.

3.1. Proposed Adaptive CNN Architecture

As illustrated in the Amazon Elastic Compute Cloud F1 services [44], having an adaptive CNN architecture that changes at runtime can secure customized service and better performance following changes in the customer's functionality and input data. One way of achieving this is by implementing CNNs as a parameterized architecture where weights, bias, number of layers, neurons per layer, and connections can be updated with the input data at runtime. Enabling dynamic model adjustments at runtime ensures the system's adaptability to changes in the environment or functional requirements. For instance, one model configuration could optimize efficiency, while an alternative configuration may prioritize accuracy [45].

We design the CNN model as a dynamic architecture (template) to be instantiated by the processing system PS every time an instantiation configuration of the template is provided. The runtime customization of the CNN model dynamically maps template parameters to the specific configuration parameters governing neurons, layers, and connections in the architecture. This mechanism ensures an efficient adaptation via instantiation. Once an instance of the CNN template is carried out every time a configuration is provided, further customization is performed via: 1) deactivation of the connections having null weights: connections between neurons that hold weights equal to zero are identified and deactivated. This eliminates unnecessary computations, as such connections do not contribute to the forward or backward propagation; 2) deactivation of Neurons with no active, outgoing connections: each neuron is analyzed for active,e outgoing connections. If all outgoing connections of a neuron are deactivated, the neuron itself becomes redundant and is subsequently deactivated. This step further reduces the computational load by removing idle neurons from the computational graph; 3) deactivation of each layer if all its neurons are deactivated: Entire layers are subject to deactivation if all neurons within the layer are deactivated due to a lack of outgoing connections. This step simplifies the model architecture by removing layers that do not contribute to the network's output.

Formally, we specify a neuron $N = \langle f, b \rangle$ through an activation function $f()$ [46] and bias $b \in \mathbb{R}$. Besides, we define a CNN layer $L = \langle N^1 \dots N^n \rangle$ as a set of neurons N^i. A CNN template T is then given by:

$$T = \langle L_1, L_2, .., L_m, C \rangle$$

where $C \in L_I \times L_I \times \mathbb{R}^+$ specifies the neuron connections given as weight coefficients. The template T is the original model architecture to be customized at runtime.

For the sake of notation, we denote the neuron N^j within layer L_i as Ni^j. Likewise, the weight of a connection between a source neuron N^s and a destination neuron N^d is represented as w_s^d. For instance, w_{12}^{21} is the weight connecting neuron N_1^2 to neuron N_2^1.

We characterize a configuration, denoted as C, as the dynamic modification to the CNN template at runtime, achieved by adjusting parameters and activation or deactivating neurons, connections, and layers. Specifically, a configuration $C \subseteq T^*$ constitutes a subset of the template, specifying values for a portion of the actual parameters. These modifications encompass alterations to the activation functions, the number of layers and neurons per layer, and connections between layers and neuron-related parameters.

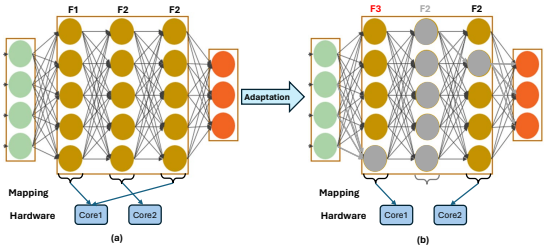


Figure 2: Example of CNN adaptation and HW reconfiguration.

For the sake of simplicity, we represent C as a function that generates a new template $T' = C(T)$ from the actual template T as stated in Eq. (1):

$$T' = T \mid \begin{cases} \forall i\ j, N_i^j \in C \implies \begin{cases} T'.L_i.N^j.f = N_i^j.f \\ T'.L_i.N^j.b = N_i^j.b \\ \forall xy\ T'.C.w_{ij}^{xy} = C.C.w_{ij}^{xy} \end{cases} \\ \forall i\ j, N_i^j \notin C \implies \begin{cases} T'.L_i.N^j.f = Neutral \\ T'.L_i.N^j.b = 0 \\ \forall x\ y\ T'.C.w_{ij}^{xy} = 0 \end{cases} \end{cases} \tag{1}$$

Fig. 2 illustrates the adaptation of a CNN template, image (**a**), to a new configuration, image (**b**), where elements greyed out (2 neurons, one layer, and many connections) are deactivated. Moreover, the activation function of the first hidden layer changed from F1 to F3. This has led to adjusting the acceleration cores allocation and active modules within the cores as explained in Section 3.2.

Recognizing the potential variability in layers, we omit explicit iteration on layers, considering it implicitly accomplished through neuron iteration. To ensure scalability for the efficient processing of large CNNs, even if the platform imposes limitations on the width of layers (number of neurons acquired and processed simultaneously), it should support the ability to partition layers into sub-layers for processing across multiple iterations. However, it is crucial to note that such a layer split necessitates the segmentation of bias vectors and weight matrices. This introduces a notable overhead in the time required to reconstruct the processed layers. It is important to acknowledge that the exploration of layer-splitting options goes beyond the scope of this article.

3.2. Proposed Acceleration Customization

Although FPGAs are limited in computation and storage resources, many recent analyses have shown that FPGAs can form a promising ground for the deployment and acceleration of future deep learning applications given the parallelization of FPGAs and the pipeline-based architecture of neural networks [47, 48]. Moreover, FPGA-based acceleration enables application flexibility and deployment optimization as explicit design steps. As an example of the potential of FPGAs, Amazon Elastic Compute Cloud (Amazon Web Services EC2) F1 instances are Xilinx FPGAs reconfigured to accelerate data workloads supporting machine learning inference [44], providing 90x higher performance than CPUs [49].

The proposed customization of acceleration cores encompasses fine-grained and coarse grained reconfiguration at runtime. The customization strategy focuses on key operational aspects to achieve seamless parallelism and reduced memory footprint. Firstly, it involves mapping layers processing from sub-images to distinct IP cores. This assignment adheres to specific guidelines such that either two layers belonging to the same sub-image are mapped to different IP cores (Intra-Sub-Image Mapping), or layers from different sub-images are allocated to different IP cores (Inter-Sub-Image Mapping), facilitating local data exchange between IP cores. This runtime arrangement promotes efficient local data exchange between IP cores, thus reducing latency and ensuring that intermediate computation results are available for subsequent processing without unnecessary communication overhead. Secondly, by monitoring the processing balance between the PS and the IP cores, the customization enables offloading an IP core if the layers' processing by a given IP core outpaces the image reassembly and reconstruction handled by the PS partition. This step ensures a balanced workflow, maintains synchronization between PS and PL partitions, and curtails the storage requirements for intermediate results, as each result produced by IP cores is immediately transferred and used by PS.

Lastly, each acceleration core is equipped with the capability to execute multiple activation functions (e.g., ReLU, Sigmoid, etc) implemented as modular components. The customization incorporates an automated mechanism to activate or deactivate these functional blocks based on the runtime CNN adaptation configuration. This adaptive approach ensures that the IP reconfiguration aligns seamlessly with the customization of the CNN template.

The capability of dynamic mapping and offloading of acceleration cores is a valuable optimization feedback mechanism for the image-splitting process [42]. It allows for dynamic adjustments to the size and quantity of sub-images per image. When PL partition demonstrates superior performance, opting for larger sub-images becomes advantageous, as it effectively diminishes the storage requirements for interim results. This reduction in storing intermediate results subsequently lowers the processing system's reassembly cost. Conversely, if the acceleration lags behind the PS performance, considering smaller sub-images and allocating layers from the same sub-image to the acceleration cores becomes a valuable choice to balance PS-PL workload and latency.

A thorough exploration of the hardware architecture and partitioning design space was undertaken to enhance the customization and performance. The findings pointed to a promising architecture: creating two hardware partitions, illustrated in Fig. 3, each containing two processing modules. This choice arises from the advantageous ability to parallelize sub-image processing and implement a pipelining approach for different functions (such as convolution and classification) within the same sub-image processing.

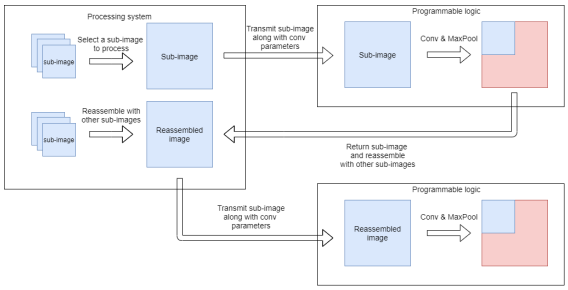


Figure 3: Multi-module architecture for PL partition

Given that input images might have large sizes due to high resolution, we consider image segmentation, where each input image is divided into smaller sub-images [50]. The partitions overlap with one pixel to preserve boundary informa-

tion, as CNN operations like convolution rely on neighborhood data. Each sub-image is passed independently through the CNN layers, performing convolution, pooling, and activation operations. The results (feature maps) for each sub-image are computed separately. To integrate the sub-images processing results, we apply image reassembling [51]. In fact, image reassembling combines the resulting sub-images to recreate a feature map corresponding to the original input image. This involves aligning the outputs correctly based on their spatial relationships in the original image. Overlapping areas between sub-images must be handled to avoid artifacts or inconsistencies in the reconstructed feature map, where methods like averaging or blending may be used to ensure a smooth transition. In our case, we adopt blending [52], where each newly integrated segment overrides a one-pixel row or column depending on the original coordinates in the input image, on each side with the already integrated segments.

To calibrate the synchronization between PS and PL partitions to minimize execution latency [53], we tune the parallelization of sub-image acceleration on PL, where either layers from the same sub-image (as tasks) execute on all IP cores or layers from different images interleave [54].

Formally, given a set of acceleration IP cores $I_1, .., I_l$, we define the latency $R(L(S), I)$ of a layer execution L to process a sub-image S on IP core I to be the time duration between the uploading of sub-image S to PL and the execution termination of L. We write $L(S)$ to refer to the processing of S by L. Thus, $R(L_m(S), I)$ refers to the response time of executing S on I since L_m is the last layer in the CNN architecture. The calibration amounts analyze the response time of the previous sub-images batch (reassembling, acceleration), compare the PS and PL performance, and adjust the parallelization of sub-images execution. The performance estimation and comparison is given by the specification in Eq.(2):

$$R(Reassemble(S_1, .., S_k)^{i_1}) > \max \sum_j (R^{i-1}(L_m(S_j), I_x)) \quad (2)$$

This constraint states that if the response time of the last sub-images reassembling in PS, from receiving the first processed sub-image S_1 to the previous one S_k, is larger than the maximum response time of the sub-images acceleration in PL, the scheduler will consider reducing the parallelization, as described later. Accordingly, at any time point t if the PS partition is faster than the PL partition, according to the performance comparison defined earlier, the scheduling of a layer L_i to process sub-image S_j is computed as described in Eq. (3):

$$Sched(L_i(S_j), t) = I_x \mid \forall y \begin{cases} R(L_i(S_j), I_x) \leq R(L_i(S_j), I_y) \\ \wedge \\ \neg\exists\, k\, l\, Sched(L_k(S_l), t) = I_y \end{cases} \quad (3)$$

Implementing this constraint decreases the interleaving of layer execution for sub-images during runtime, subsequently enhancing latency. This reduction allows for the storage of fewer intermediate results and minimizes the utilization of AXI buses, ultimately contributing to improved execution performance. The control module of the IPs dynamically adjusts the allocation of layers and sub-images based on inputs from the Processing System (PS) and adherence to the constraints above. Parallel processing from different sub-images can be increased, and real-time random computation of hardware allocation is facilitated to achieve a more efficient mapping with reduced latency as given Eq. (4):

$$Sched(L_i(S_j), t) = I_x \mid \forall y R(L_i(S_j), I_x) \leq R(L_i(S_j), I_y) \quad (4)$$

Lastly, an initial static binding of the template layers to the IP cores is established, which will be dynamically adjusted at runtime based on the computational load of adaptive CNN layers. The computation load of a given layer, denoted as L_i, is quantified by the computation cost of its activation functions, denoted as $|\,|$, benchmarked on the target FPGA board [55], as stated in Eq. (5):

$$load(L_i) = \sum_1^{n_i} |N_i.f| \quad (5)$$

Accordingly, as given in Eq. (6), a layer is defined to be *NEUTRAL* if it has an empty computation load, i.e., to simulate an inactive layer where all activation functions are neutral.

$$NEUTRAL(L_i) = \{\forall j\, N_i^j.f = Neutral\} \quad (6)$$

At runtime, when a layer is excluded from the CNN template due to having a neutral load in the adaptive architecture, the original IP core designated to execute that excluded layer, denoted as L_i, is repurposed to execute the subsequent layer in the sequence. This is to avoid computing an entirely new schedule. Meanwhile, the outputs presumed to be generated by layer Li are utilized as inputs for the next layer in the CNN architecture. Consequently, this dynamic adjustment in the runtime schedule ensures that the next layer, denoted as $Li + 1$, is scheduled for execution using the IP core initially assigned to L_i. Namely, the scheduling is update is given in Eq. (7).

$$Sched(L_i, t) = -1 \text{ if } NEUTRAL(L_i) \quad (7)$$

Accordingly, whenever a layer L_i becomes neutral, the scheduling of the next layers L_{i+z}, with $z \in \{1, m - i\}$ will be updated as stated in Eq. (8).

$$Sched(L_{i+z}, t) = Sched(L_{i+z-1}, t - 1) \quad (8)$$

The schedule update is iterative, so whenever a layer is deactivated, customization runs through mapping all layers and updating it accordingly.

4. Implementation and Performance Evaluation

This section elaborates on the implementation and experiments and compares the results to the state-of-the-art, with respect to accuracy, computation performance, hardware utilization, and scalability.

4.1. Implementation

The implementation and testing of the proposed methodology utilized the rapid prototyping PYNQ framework, as it allows for work at a higher abstraction when interacting with the acceleration IP cores, where those cores are wrapped as functions to call from the application code in PS. This is particularly efficient when carrying out design space exploration on Xilinx MPSoCs. PYNQ further enabled easy memory allocation in the DDR RAM on the platform, making intermediate storage of weights, biases, and fmaps possible.

The original implementation was made in Python due to the availability of tools and open-source libraries for image processing and training the CNN network. Namely, Keras has been used to create the CNN template, train and test it on the public data sets MNIST and CODaN. Keras further enables easier storage of the CNN models as it offers many different formats to store the networks, where, in this case, the *h5* format is used [56]. The training outcomes are then supplied to the CNN template on the FPGA through an SD card for runtime customization. The runtime customization configuration of the CNN can be triggered upon reading the SD card or can as well be time-triggered.

It was chosen to synthesize many independent IP cores. Each IP core was created using Vitis HLS, where the IP core was implemented, optimized using pragma, tested, synthesized, and exported to Vivado to integrate later with the PS partition code. Furthermore, in the implementation of the CNN template T we considered the following: $m \leq 6$, $n \leq 512$ and $\sum_{i=1}^{m} n_i \leq 2048$. When the current image block layers have been processed, the CNN controller triggers a callback where the next layers are determined and sent to the corresponding IP core set in the Layer Controller. The implementation code, including CNN test data, high-level synthesis of the hardware accelerators, and application integration, is available here [1].

4.2. Experiments

We have conducted a large set of experiments to analyze the performance, accuracy, resource utilization, memory storage, and scalability of the proposed acceleration architecture. We have considered the following parameters to define the different experiments: sub-image resolution (splitting size), activation functions, CNN depth (number of layers), number of neurons per layer, and quantization size to represent and store the CNN parameters and data. For each experiment, we maintain all the parameters constant and only vary one at a time. Table 1 summarizes the experiment sets we conducted where the variable parameters are highlighted in parenthesis. In total, 24 experiments were carried out with more than 56 analyses, and for each experiment, we assessed accuracy, resource utilization, and execution performance.

4.3. Results and Discussion

Accuracy Analysis. The accuracy analysis was conducted on MNIST and CODaN datasets with variable parameters. Fig. 4a presents the accuracy loss for the MNIST data set while varying the image split size. Our model demonstrates an accuracy range of 93% to 97% in both the 14 and 28 splitting experiments. This range falls within the acceptable threshold compared to state-of-the-art analyses utilizing the same datasets. For instance, previous studies such as [57] reported an accuracy of 93%, while [58] achieved an accuracy of 96%. We observed that the accuracy level is the same when a convolutional block of one and two is used. However, a minimal accuracy loss is introduced when the length of the convolution blocks is larger than two. Similarly, an accuracy analysis is conducted on the CODaN dataset. Fig. 4b shows the accuracy results where a higher accuracy loss is introduced due to data being much larger and diverse compared to MNIST.

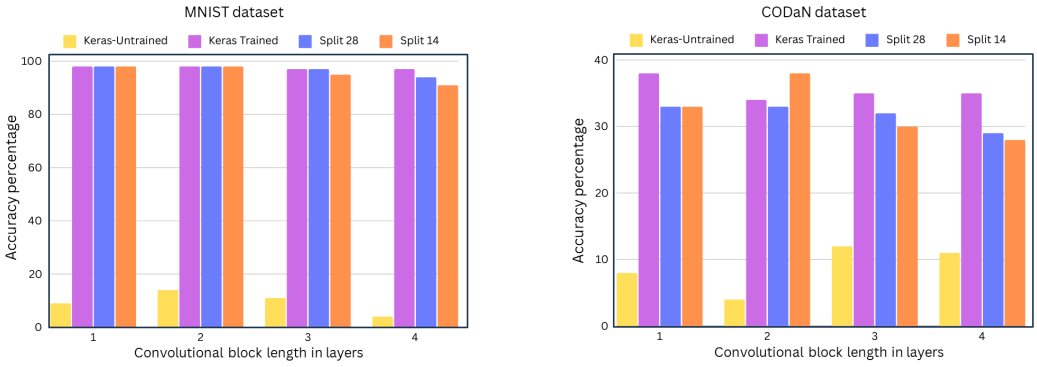
Table 1: Summary of the experiments and tests performed.

Exp \ Test values	test 1	test 2	test 3	test 4	test 5	test 6
Experiment set1 (sub-image size)	12*12	18*18	24*24	28*28	34*34	52*52
Experiment set2 (layers number)	3	4	5	6	7	8
Experiment set3 (neurons per layer)	32	64	128	256	512	1024
Experiment set4 (quantization)	Q(2,14)	Q(4,4)	Q(8,8)	Q(16,16)	Q(32,16)	Q(32,32)

Computation Performance Analysis. In Fig. 5a, the comprehensive analysis of computation performance depicts the total execution time and the acceleration time (hardware time) required for one adaptation of the CNN architecture and the processing of 10 images. In the best case (Split28), our proposed acceleration environment processes 10 images in 18.5s, including initial *parsing* of the CNN architecture, image *partitioning*, *reassembly*, *classification* and another CNN parsing via *adaptation*. A breakdown analysis of the execution reveals that up to 53% of the 18.5s duration is used to read and parse both the initial and the adaptation CNN configurations, each consisting of at least 140000 parameters from the SD card, and up to 10% to fetch the input images from the SD card. Thus, the actual computation time to process 10 images with a split of 28 is 6.8s, with an average of **0.68s** per image. The experiment employing a 14x14 split executes in 68 seconds, while its counterpart with a 28x28 split completes the processing in 18.5 seconds. One can observe that a significant portion, approximately 80%, of the total execution time for the 14x14 split is consumed outside the hardware accelerators to fetch and store the high number of sub-images [18]. In fact, the high processing time in PS is due to parsing the CNN template (from an off-chip memory), computing new schedules to allocate the acceleration cores, segmentation, and storage of the increased number of 14x14 sub-images. This leads to heightened overhead time in PS to reassemble processed sub-images and a higher frequency of sending and retrieving sub-images between PS and PL partitions. By applying a split of 52x52 (test case 6), the total processing time for the same experiment as above converges to 1 second.

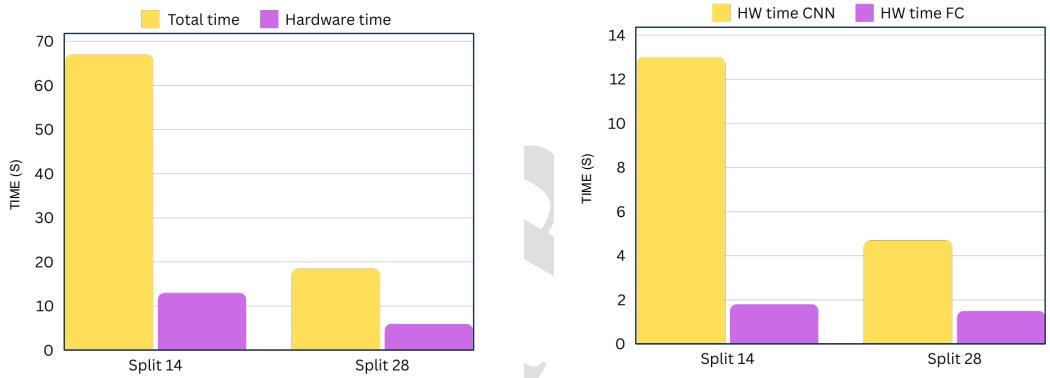
Since we have synthesized mainly two different IP core classes, one for classification and one for convolution, we analyzed the execution time of both hardware accelerators as depicted in

[1] https://e.pcloud.link/publink/show?code=XZvDN2ZW2YSsz7tHMQzevCkcOojnVArsSiX



(a) (b)

Figure 4: Accuracy analysis on MNIST and CODaN datasets.



(a) (b)

Figure 5: Analysis of execution and acceleration time.

Fig. 5b. It can be seen that the time spent performing CNN convolutions is largely higher than the classification time. This is, in fact, due to convolution being applied to all intermediate sub-images, whereas classification is executed only once on the final (reassembled) image.

Hardware Utilization and Scalability analysis. Considering that CNN size and image resolution significantly influence the overall resource utilization and scalability, we conducted various analyses by adjusting the total number of CNN neurons and pixels per image.

In Fig. 6, the utilization of hardware fabric logic resources, including RAM blocks (BRAM), digital signal processors (DSP), flip-flops (FF), and look-up tables (LUT), is depicted for the CNN architectures ranging from 32 to 2048 neurons. Notably, storage requirements exhibit a quadratic increase with input size expansion. Conversely, DSPs, FFs, and LUTs display linear growth, as these resources are statically determined by the number of neurons rather than the input size. Consequently, BRAM capacity may present a bottleneck for deploying CNNs with more neurons. Given that the board we use contains 230K LUT, the acceleration core can be scaled up to incorporate and process up to 2700 Neurons in parallel however, this bottleneck can be bypassed by reusing IP core circuits and serializing the execution of some of the neurons although this can slow down execution pace. Furthermore, typical neural networks contain far less than that large number of neurons per layer.

Fig. 7 depicts the hardware resource utilization following the input image size. One can see that BRAM and URAM have a linear complexity relative to the input image size, whereas DSPs, FFs, and LUTs do not demonstrate any specific increase pattern. This might require further investigation to identify a particular dependency pattern so that deployment feasibility can be assessed early enough for the input image sizes. However, it is important to state that the input image size does not repre-

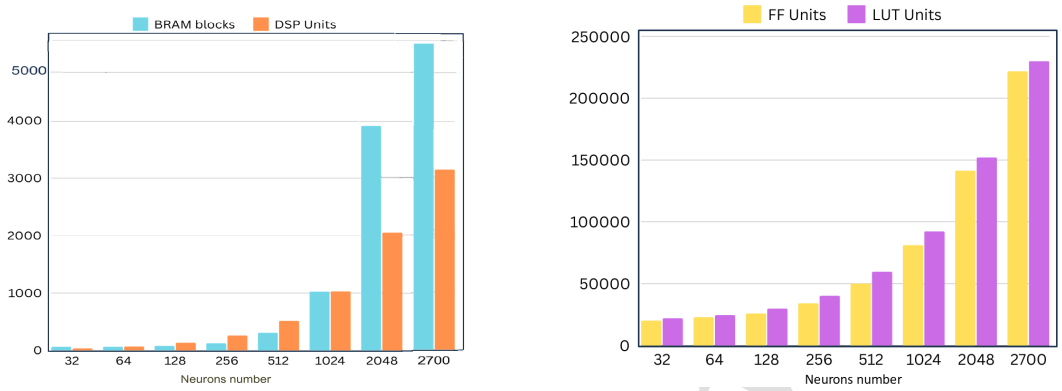


Figure 6: PL resources utilization and scalability.

sent a deployment bottleneck given the modular processing of images via splitting. Thus, high-resolution images can be processed in a similar way via a larger number of splitting and reassembling operations. Indeed, the number of splits to perform depends on the maximum input size of the IP core adopted.

Since the customization involves the activation and deactivation of acceleration core modules, as each IP is synthesized to execute functions such as convolution, classification, and computation utilizing distinct activation functions, we have observed that if the current layer size is less than 50% of the original template layer size employed during IP synthesis, up to 35% of the IP circuits remain inactive.

Comparison to the state-of-the-art. As stated earlier, the proposed acceleration framework outperforms different state-of-the-art studies in terms of accuracy [60, 57]. Moreover, the achieved computation performance (0.68s/image) outperforms the computation performance of 2.5s/image and 2.19s/image achieved in [36] and [62], respectively. This is, in fact, due to the parallelization of the sub-images processing, with a dynamic overlapping (intra- and inter-subimage mapping), and the efficient load balancing between PS and PL partitions. Thanks to our efficient implementation, parallelization efficiency is not achieved at the expense of large hardware sizes. Rather, the IP components are re-utilized for sub-image processing, whereas the *Pipeline* directive executes the for-loops within each sub-image processing. Table 2 summarizes a comparison to the relevant state-of-the-art studies by considering classification accuracy, computation performance, and the number of hardware resources used for acceleration.

One can see that while delivering the highest accuracy and moderate computation performance, the proposed acceleration requires one of the *lowest* hardware resources set for acceleration. This will result in high scalability to accelerate larger CNN architectures and achieve high energy efficiency.

5. Conclusions and Future Work

This article developed a methodology for deploying adaptable, scalable, and hardware-accelerated convolutional neural networks on an embedded platform for image processing applications. The proposed architecture facilitates CNNs' runtime adaptability and dynamic configuration of the hardware accelerators to leverage execution performance. The innovation lies in an iteration-free synthesis approach, where hardware IPs are synthesized once and configured at runtime following the CNN architecture, significantly reducing design and deployment costs.

A prototype was implemented in Python to validate the feasibility of the proposed acceleration and deployment processes. This prototype enabled CNN training and parameter generation using Keras. Vitis HLS was employed to synthesize and optimize hardware accelerators on a Xilinx FPGA board, while the PYNQ environment integrated the software application with the synthesized hardware accelerators.

Extensive experiments, encompassing datasets such as MNIST and CODaN with up to 180,000 parameters, were conducted to evaluate the execution performance, accuracy, resource utilization, and scalability. The results indicate that our prototype surpasses the state-of-the-art in terms of accuracy and deployment cost, especially when changes to the CNN architecture or functionality do not necessitate IP core synthesis.

In the future, we aim to enhance adaptability by incorporating a broader range of activation functions and strive for better alignment of computation loads across IP accelerators to optimize response times. Additionally, automated inference of optimized configurations during initial hardware synthesis will be crucial for further development. It is also worth investigating efficient on-chip memory utilization to reduce the off-chip bottleneck and improve the latency further.

References

[1] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, B. Hodjat, Evolving deep

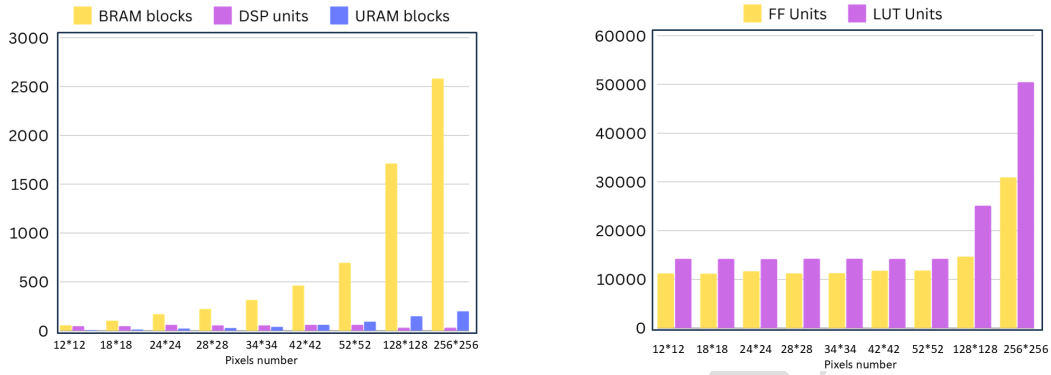


Figure 7: PL resources utilization following input image size.

Table 2: Comparison to the state-of-the-art.

Ref	Accuracy (%)	Latency (s)	FF	LUT	DSP	BRAM
Luo et al. [59]	95	0.33	49K	49K	44	93
Wen et al. [60]	94	0.035	25K	20K	576	149
Waseem et al. [61]	-	0.11	43K	17K	25	173
Wang et al. [62]	89	2.19	131K	181K	576	435
Huang et al. [36]	-	2.5	-	-	-	-
Meshkini et al. [57]	93	-	-	-	-	-
Proposed Approach	**97**	**0.68**	**11K**	**14K**	**54**	**178**

neural networks (2024).
[2] A. Haleem, M. Javaid, M. Asim Qadri, R. Pratap Singh, R. Suman, Artificial intelligence (ai) applications for marketing: A literature-based study, International Journal of Intelligent Networks 3 (2022) 119–132.
[3] Z. Wang, M. Goudarzi, M. Gong, R. Buyya, Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments, Vol. 152, 2024, pp. 55–69.
[4] A. Khdoudi, T. Masrour, I. El Hassani, C. El Mazgualdi, A deep-reinforcement-learning-based digital twin for manufacturing process optimization, Systems 12 (2) (2024) 38.
[5] M. Rafiei, J. Boudjadar, M. P. Griffiths, M.-H. Khooban, Deep learning-based energy management of an all-electric city bus with wireless power transfer, IEEE Access 9 (2021) 43981–43990.
[6] B. Saleem, R. Badar, A. Manzoor, M. A. Judge, J. Boudjadar, S. U. Islam, Fully adaptive recurrent neuro-fuzzy control for power system stability enhancement in multi machine system, IEEE Access 10 (2022) 36464–36476.
[7] N. Hu, D. Zhang, K. Xie, W. Liang, K.-C. Li, A. Y. Zomaya, Dynamic multi-scale spatial–temporal graph convolutional network for traffic flow prediction, Future Generation Computer Systems 158 (2024) 323–332.
[8] P. W. McCulloch WS, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics 52 (1943) 115–133.
[9] P. Freire, S. Srivallapanondh, B. Spinnler, A. Napoli, N. Costa, J. E. Prilepsky, S. K. Turitsyn, Computational complexity optimization of neural network-based equalizers in digital signal processing: A comprehensive approach, Journal of Lightwave Technology 42 (12) (2024) 4177–4201.
[10] Y. Wang, Y. Han, C. Wang, S. Song, Q. Tian, G. Huang, Computation-efficient deep learning for computer vision: A survey, Cybernetics and Intelligence (2024).
[11] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, B. Hodjat, Evolving deep neural networks, in: Artificial Intelligence in the Age of Neural Networks and Brain Computing (Second Edition), Academic Press USA (2024), pp. 269–287.
[12] M. Yasir, S. Liu, X. Mingming, J. Wan, S. Pirasteh, K. B. Dang, Shipgeonet: Sar image-based geometric feature extraction of ships using convolutional neural networks, IEEE Transactions on Geoscience and Remote Sensing (2024).
[13] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, M. Parmar, A review of convolutional neural networks in computer vision, Artificial Intelligence Review 57 (4) (2024).
[14] Y. Zhang, A survey on evaluation methods for image segmentation, Pattern Recognition 29 (8) (1996) 1335–1346.
[15] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, S. Almotairi, A comparison of pooling methods for convolutional neural networks, Applied Sciences 12 (17) (2022).
[16] G. D. Licciardo, C. Cappetta, L. Di Benedetto, Design of a convolutional two-dimensional filter in fpga for image processing applications, Computers 6 (2) (2017).
[17] P. Grzesik, D. Mrozek, Combining machine learning and edge computing: Opportunities, challenges, platforms, frameworks, and use cases, Electronics 13 (3) (2024) 640.
[18] Y. Wan, X. Xie, J. Chen, K. Xie, D. Yi, Y. Lu, K. Gai, Ads-cnn: Adaptive dataflow scheduling for lightweight cnn accelerator on fpgas, Future Generation Computer Systems 158 (2024) 138–149.
[19] S. Bertazzoni, L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, M. Re, S. Spanò, Design space exploration for edge machine learning featured by mathworks fpga dl processor: A survey, IEEE Access (2024).
[20] P. Kumar, I. Ali, D.-G. Kim, S.-J. Byun, D.-G. Kim, Y.-G. Pu, K.-Y. Lee, A study on the design procedure of re-configurable convolutional neural network engine for fpga-based applications, Electronics 11 (23) (2022).
[21] J. a. G. Reis, A. A. Fröhlich, Towards deterministic fpga reconfiguration, International Journal of Embedded Systems 13 (2) (2020) 236–253.
[22] Y. Yang, C. Wang, X. Zhou, Drama: A high efficient neural network accelerator on fpga using dynamic reconfiguration: Work-in-progress, in: Proceedings of Intl. Conf. on Hardware/Software Codesign and System Synthesis Companion, 2019.
[23] M. Sponner, B. Waschneck, A. Kumar, Adapting neural networks at runtime: Current trends in at-runtime optimizations for deep learning, ACM

Comput. Surv. 56 (10) (2024).
[24] A. Dimitriou, B. Biggs, J. Hare, G. V. Merrett, Fpga acceleration of dynamic neural networks: Challenges and advancements, in: 2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS), 2024.
[25] E. Jaballi, S. Gdaim, N. Liouane, Design and implementation of fpga-based hardware acceleration for machine learning using opencl: A case study on the k-means algorithm, in: 2024 International Conference on Control, Automation and Diagnosis (ICCAD), 2024.
[26] A. El Bouazzaoui, A. Hadjoudja, O. Mouhib, N. Cherkaoui, Fpga-based ml adaptive accelerator: A partial reconfiguration approach for optimized ml accelerator utilization, Array 21 (2024).
[27] K. P. Seng, P. J. Lee, L. M. Ang, Embedded intelligence on fpga: Survey, applications and challenges, Electronics 10 (2021).
[28] K. Shi, M. Wang, X. Tan, Q. Li, T. Lei, Efficient dynamic reconfigurable cnn accelerator for edge intelligence computing on fpga, Information 14 (3) (2023).
[29] T. S. Ajani, A. L. Imoize, A. A. Atayero, An overview of machine learning within embedded and mobile devices–optimizations and applications, Sensors 21 (13) (2021) 4412.
[30] A. Fanariotis, T. Orphanoudakis, K. Kotrotsios, V. Fotopoulos, G. Keramidas, P. Karkazis, Power efficient machine learning models deployment on edge iot devices, Sensors 23 (2023).
[31] I. Kuon, R. Tessier, J. Rose, Fpga architecture: Survey and challenges, Foundations and Trends® in Electronic Design Automation 2 (2) (2008) 135–253.
[32] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, Z. Zhang, High-level synthesis for fpgas: From prototyping to deployment, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (4) (2011) 473–491.
[33] S. Branco, A. G. Ferreira, J. Cabral, Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey, Electronics 8 (11) (2019) 1289.
[34] J. Jakobsen, M. Jensen, I. Sharifirad, J. Boudjadar, A flexible implementation model for neural networks on fpgas, in: International Conference on Intelligent Systems Design and Applications, Springer, 2022, pp. 332–342.
[35] F. Ratto, Á. P. Máinez, C. Sau, P. Meloni, G. Deriu, S. Delucchi, M. Massa, L. Raffo, F. Palumbo, An automated design flow for adaptive neural network hardware accelerators, Journal of Signal Processing Systems 95 (9) (2023) 1091–1113.
[36] C.-H. Huang, S.-W. Tang, P.-A. Hsiung, Acnne: An adaptive convolution engine for cnns acceleration exploiting partial reconfiguration on fpgas, in: 2024 IEEE International Symposium on Circuits and Systems (ISCAS), 2024.
[37] H. Tong, K. Han, S. Han, Y. Luo, Design of a generic dynamically reconfigurable convolutional neural network accelerator with optimal balance, Electronics 13 (2024).
[38] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, X. Zhou, Dlau: A scalable deep learning accelerator unit on fpga, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36 (3) (2016) 513–517.
[39] K. Neshatpour, H. M. Mokrani, A. Sasan, H. Ghasemzadeh, S. Rafatirad, H. Homayoun, Architectural considerations for fpga acceleration of machine learning applications in mapreduce, in: Proceedings of Intl. Con. on embedded computer systems: Architectures, modeling, and simulation, 2018, pp. 89–96.
[40] V. Gokhale, A. Zaidy, A. X. M. Chang, E. Culurciello, Snowflake: An efficient hardware accelerator for convolutional neural networks, in: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017.
[41] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, L. Benini, Neuraghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs, ACM Trans. Reconfigurable Technol. Syst. 11 (3) (2018).
[42] S. I. Venieris, C.-S. Bouganis, fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas, IEEE Transactions on Neural Networks and Learning Systems 30 (2) (2019).
[43] C. Gao, F. Zhang, Fpga-based accelerator for independently recurrent neural network, in: 2018 IEEE 4th International Conference on Computer and Communications (ICCC), 2018.
[44] AMD Xilinx, Aws cloud: Xilinx fpgas in world's largest cloud, `https://www.xilinx.com/products/design-tools/acceleration-zone/aws.html`.
[45] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, arXiv preprint arXiv:1608.08710 (2016).
[46] S. Qian, H. Liu, C. Liu, S. Wu, H. S. Wong, Adaptive activation functions in convolutional neural networks, Neurocomputing 272 (2018) 204–212.
[47] A. Nechi, L. Groth, S. Mulhem, F. Merchant, R. Buchty, M. Berekovic, Fpga-based deep learning inference accelerators: Where are we standing?, ACM Trans. Reconfigurable Technol. Syst. 16 (4) (2023).
[48] A. G. Blaiech, K. B. Khalifa, C. Valderrama, M. A. Fernandes, M. H. Bedoui, A survey and taxonomy of fpga-based deep learning accelerators, Journal of Systems Architecture 98 (2019) 331–345.
[49] AMD Xilinx, `https://www.xilinx.com/applications/megatrends/machine-learning.html`.
[50] S. Bose, A. Mukherjee, Madhulika, S. Chakraborty, S. Samanta, N. Dey, Parallel image segmentation using multi-threading and k-means algorithm, in: 2013 IEEE International Conference on Computational Intelligence and Computing Research, 2013.
[51] T. Yan, G. Chen, H. Zhang, G. Wang, Z. Yan, Y. Li, S. Xu, Q. Zhou, R. Shi, Z. Tian, B. Wang, Convolutional neural network with parallel convolution scale attention module and rescbam for breast histology image classification, Heliyon 10 (2024).
[52] C.-N. Lu, Y.-C. Chang, W.-C. Chiu, Bridging the Visual Gap: Wide-Range Image Blending , in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.
[53] J. Boudjadar, S. Ramanathan, A. Easwaran, U. Nyman, Combining task-level and system-level scheduling modes for mixed criticality systems, in: 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2019.
[54] B. Madzar, J. Boudjadar, J. Dingel, T. E. Fuhrman, S. Ramesh, Formal analysis of predictable data flow in fault-tolerant multicore systems, in: Formal Aspects of Component Software, 2017.
[55] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, H. Esmaeilzadeh, From high-level deep neural models to fpgas, in: Intl. Symposium on Microarchitecture (MICRO), 2016.
[56] The HDF Group, `https://www.hdfgroup.org/solutions/hdf5/` (2022).
[57] K. Meshkini, J. Platos, H. Ghassemain, An analysis of convolutional neural network for fashion images classification (fashion-mnist), in: Proceedings of Conf. on Intelligent Information Technologies for Industry, 2020.
[58] K. Cheng, R. Tahir, L. K. Eric, M. Li, An analysis of generative adversarial networks and variants for image synthesis on mnist dataset, Multimed Tools Appl 79 (2020) 13725–13752.
[59] Y. Luo, X. Cai, J. Qi, D. Guo, W. Che, Fpga–accelerated cnn for real-time plant disease identification, Computers and Electronics in Agriculture 207 (2023).
[60] L. C. Hongxing Wen, Software and hardware synergy for accelerated plant disease identification, Applied Soft Computing 61 (2024).
[61] S. M. Waseem, S. K. Roy, Chapter 10 - fully convolutional network for edge devices—fpga implementation and analysis for agriculture technology, in: Agri 4.0 and the Future of Cyber-Physical Agricultural Systems, Academic Press (2024), 2024, pp. 175–196.
[62] X. Wang, Z. Zhou, Z. Yuan, J. Zhu, Y. Cao, Y. Zhang, K. Sun, G. Sun, Fd-cnn: A frequency-domain fpga acceleration scheme for cnn-based image-processing applications, ACM Trans. Embed. Comput. Syst. 22 (2023).

Jalil Boudjadar Photo

Click here to access/download;Author Photo;Jalil Boudjadar Picture.jpg

Saif Ul Islam Photo

Click here to access/download;Author Photo;Saif ul Islam Picture.jpg

Rajkumar Buyya Photo

Click here to access/download;Author Photo;Rajkumar Buyya Picture.jfif

Highlights

Paper Title: Dynamic FPGA Reconfiguration for Scalable Embedded Artificial Intelligence (AI): A Co-Design Methodology for CNN Acceleration

1. Novel co-design framework that integrates adaptive CNNs with FPGA platforms to eliminate iterative synthesis, reducing deployment time and costs.
2. FPGA hardware can be reconfigured at runtime to adapt CNN architectures and hyperparameters without full re-synthesis.
3. Combines fine- and coarse-grained hardware customizations, improving computation performance and reducing latency.
4. Tested on Xilinx ZYBO FPGA with a CNN-based image classifier, demonstrating improved efficiency and scalability.
5. An adaptable, scalable solution for deploying AI applications on FPGA platforms through dynamic reconfiguration.

Authors' Biographies

Jalil Boudjadar received the M.Sc. degree from Limoges University in June 2008 and the Ph.D. degree from Toulouse University, France, in December 2012. He is currently an associate professor in the Electrical and Computer Engineering (ECE) Section of the Department of Engineering. He is also a member of the DIGIT Research Centre. His research interests include the design, validation, and optimization of embedded systems; modeling and intelligent control for cyber-physical systems; digitization, digital twins, and adaptive runtime engineering; machine learning and embodied artificial intelligence; real-time systems and scheduling theories; and formal methods for security, safety, and data analysis.

Saif Ul Islam received a Ph.D. degree in distributed systems from Université Toulouse III Paul Sabatier, Toulouse, France, in 2015 under the supervision of Prof. Jean-Marc Pierson, the head of Institut de Recherche en Informatique de Toulouse (IRIT). He is currently a Research Fellow at WMG, University of Warwick, UK. Previously, he was an Associate Professor with the Department of Computer Science, Institute of Space Technology, Islamabad, Pakistan, and an Assistant Professor with COMSATS University, Islamabad, Pakistan. He is currently working and has been part of European Union-funded research projects. He has published his research in various reputed journals. He has an H-Index of 31 at Google Scholar. He is the associate editor of two journals and also guest-editing a few special issues in different journals. His research interests include parallel and distributed computing, sustainable computing, artificial intelligence, fog computing, edge computing, and machine learning.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercialising its innovations in Cloud Computing. He served as a Future Fellow of the Australian Research Council during 2012-2016. He serving/served as Honorary/Visiting Professor for several elite Universities including Imperial College London (UK), University of Birmingham (UK), University of Hyderabad (India), and Tsinghua University (China). He received B.E and M.E in Computer Science and Engineering from Mysore and Bangalore Universities in 1992 and 1995 respectively; and a Doctor of Philosophy (PhD) in Computer Science and Software Engineering from Monash University, Melbourne, Australia in 2002. He was awarded Dharma Ratnakara Memorial Trust Gold Medal in 1992 for his academic excellence at the University of Mysore, India. He received Richard Merwin Award from the IEEE Computer Society (USA) for excellence in academic achievement and professional efforts in 1999. He received Leadership and Service Excellence Awards from the IEEE/ACM International Conference on High Performance Computing in 2000 and 2003. He received "Research Excellence Awards" from the University of Melbourne for productive and quality research in computer science and software engineering in 2005 and 2008. He acknowledges all researchers and institutions worldwide for their consideration in building on software systems created by his CLOUDS Lab and recognising them through citations and contributing to their further enhancements. With over 152,000 citations, a g-index of 371, and an h-index of 168, he is one of the highly cited authors in computer science and software engineering worldwide. He received the Chris Wallace Award for Outstanding Research Contribution 2008 from the Computing Research and Education Association of Australasia, CORE, which is an association of university departments of computer science in Australia and New Zealand. Dr. Buyya received the "2009 IEEE TCSC Medal for Excellence in Scalable Computing" for pioneering the economic paradigm for utility-oriented distributed computing platforms such as Grids and Clouds. He served as the founding Editor-in-Chief (EiC) of IEEE Transactions on Cloud Computing (TCC). Dr.

Buyya is recognized as a "Web of Science Highly Cited Researcher" for seven times since 2016, Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier, and "Lifetime Achievement Awards" from two Indian universities for his outstanding contributions to Cloud computing and distributed systems. He has been recognised as the "Best of the World" twice for research fields (in Computing Systems in 2019 and Software Systems in 2021) as well as "Lifetime Achiever" and "Superstar of Research" in "Engineering and Computer Science" discipline twice (2019 and 2021) by the Australian Research Review. Recently, he received "Research Innovation Award" from IEEE Technical Committee on Services Computing and "Research Impact Award" from IEEE Technical Committee on Cloud Computing.

Dr. Buyya has contributed to the creation of high-performance computing and communication system software for PARAM supercomputers developed by the Centre for Development of Advanced Computing (C-DAC), India. He has pioneered Economic Paradigm for Service-Oriented Distributed Computing and demonstrated its utility through his contribution to conceptualisation, design and development of Grid and Cloud Computing technologies such as Aneka, GridSim, Libra, Nimrod-G, Gridbus, and Cloudbus that power the emerging eScience and eBusiness applications. He has been awarded, over $8 million, competitive research grants from various national and international organisations including the Australian Research Council (ARC), Sun Microsystems, StorageTek, IBM, and Microsoft, CA Australia, Australian Dept. of Innovation, Industry, Science and Research (DIISR), and European Council. Dr. Buyya has been remarkably productive in a research sense and has converted much of that knowledge into linkages with industry partners (such as IBM, Sun and Microsoft), into software tools useful to other researchers in a variety of scientific fields, and into community endeavours. Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 50+ countries around the world. In recognition of this, he received Vice Chancellor's inaugural "Knowledge Transfer Excellence (Commendation) Award" from the University of Melbourne in Nov 2007. Manjrasoft's Aneka technology for Cloud Computing developed under Dr.Buyya's leadership has received "2010 Asia Pacific Frost & Sullivan New Product Innovation Award". Recently, Dr. Buyya received "Bharath Nirman Award" and "Mahatma Gandhi Award" along with Gold Medals for his outstanding and extraordinary achievements in Information Technology field and services rendered to promote greater friendship and India-International cooperation.

Dr. Buyya has authored/co-authored over 850 publications. Since 2007, he received twelve "Best Paper Awards" from international conferences/journals including a "2009 Outstanding Journal Paper Award" from the IEEE Communications Society, USA. He has co-authored five text books: *Microprocessor x86 Programming* (BPB Press, New Delhi, India, 1995), *Mastering C++* (McGraw Hill Press, India, 1st edition in 1997 and 2nd edition in 2013), *Object Oriented Programming with Java: Essentials and Applications* (McGraw Hill, India, 2009), *Mastering Cloud Computing* (Morgan Kaufmann, USA; McGraw Hill, India, 2013; China Machine Press, 2015), and *Cloud Data Centers and Cost Modeling* (Morgan Kaufmann, USA, 2015). The books on emerging topics that he edited include, *High Performance Cluster Computing* (Prentice Hall, USA, 1999), High Performance Mass Storage and Parallel I/O (IEEE and Wiley Press, USA, 2001), Content Delivery Networks (Springer, Germany, 2008), Market Oriented Grid and Utility Computing (Wiley Press, USA, 2009), and Cloud Computing: Principles and Paradigms (Wiley, USA, 2011). He also edited proceedings of over 25 international conferences published by prestigious organisations, namely the IEEE Computer Society Press (USA) and Springer Verlag (Germany). He served as Associate Editor of Elsevier's Future Generation Computer Systems Journal (2004-2009) and currently serving on editorial boards of many journals including Software: Practice and Experience (Wiley Press). Dr. Buyya served as a speaker in the IEEE Computer Society Chapter Tutorials Program (from 1999-2001), Founding Co-Chair of the IEEE Task

Force on Cluster Computing (TFCC) from 1999-2004, and member of the Executive Committee of the IEEE Technical Committee on Parallel Processing (TCPP) from 2003-2011. He served as the first elected Chair of the IEEE Technical Committee on Scalable Computing (TCSC) during 2005-2007 and played a prominent role in the creation and execution of several innovative community programs that propelled TCSC into one of the most successful TCs within the IEEE Computer Society. In recognition of these dedicated services to computing community over a decade, President of the IEEE Computer Society presented Dr. Buyya a Distinguished Service Award in 2008.

Dr. Buyya is a Fellow of IEEE, Foreign Fellow of Academia Europaea, and Life Member of ACM. He has co-founded five IEEE/ACM international conferences: CCGrid, Cluster, Grid, e-Science, and UCC (Utility and Cloud Computing) and served as the Chair of their inaugural meetings. He served as a Member of the IEEE Computer Society Fellow Evaluating Committee in 2015, 2018, and 2021. He has presented over 600 invited talks (keynotes, tutorials, and seminars) on his vision on IT Futures and advanced computing technologies at international conferences and institutions in Asia, Australia, Europe, North America, and South America. For further information on Dr. Buyya, please visit: http://www.buyya.com

Author Names and their Contribution

Authors' Contributions

Paper Title: Dynamic FPGA Reconfiguration for Scalable Embedded Artificial Intelligence (AI): A Co-Design Methodology for CNN Acceleration

Jalil Boudjadar: Conceptualization, Methodology, Software, Implementation, Writing - Original Draft

Saif Ul Islam: Validation, Analysis, Project Management, Writing - Review & Editing

Rajkumar Buyya: Supervision, Validation, Writing - Review & Editing

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The author is an Editorial Board Member/Editor-in-Chief/Associate Editor/Guest Editor for *[Journal name]* and was not involved in the editorial review or the decision to publish this article.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: