Signal, Image and Video Processing (2024) 18 (Suppl 1):S599–S615
https://doi.org/10.1007/s11760-024-03177-2

ORIGINAL PAPER

Fpga-based SoC design for real-time facial point detection using deep convolutional neural networks with dynamic partial reconfiguration

Safa Teboulbi[1] · Seifeddine Messaoud[2] · Mohamed Ali Hajjaji[3] · Abdellatif Mtibaa[4] · Mohamed Atri[5]

Received: 6 February 2024 / Revised: 16 March 2024 / Accepted: 22 March 2024 / Published online: 14 May 2024

Abstract
Deep convolutional neural networks (DCNNs) have been mainly powerful and important artificial intelligence techniques, which are exploited in various computer vision applications, such as facial point detection (FPD), owing their versatility and high performance. The role of DCNNs in this area is pivotal due to their ability to learn hierarchical features, which are essential for recognizing complex patterns. On the other hand, DCNNs have considerable computational complexity due to their topology nature. An FPGA-based SoC design is needed for CNN acceleration due to the rapid development, energy efficiency, low latency and high reconfigurability of FPGAs. Despite their performance benefits, FPGA-based SoCs also come with several limitations like complexity of design. This paper proposes a dynamic partial reconfiguration (DPR) and hybrid architecture for DCNN accelerators. We propose (i) a GPU-based software implementation for DCNN-based FPD, (ii) a CNN-based acceleration and optimization method using the high-level synthesis technique, (iii) a DPR-based hybrid architecture to improve the performance of the suggested approach. To validate our design, four scenarios are put forward. The experimental results prove that the proposed work achieves a better performance in terms of reconfigurability and execution time, hardware cost, and power consumption.

Keywords CNN · FPGA-SoC · Acceleration · Dynamic reconfiguration · Co-design

1 Introduction

Real-time facial point detection (FPD) refers to the process of identifying and tracking specific key points (or landmarks) on a face in real time. These points typically include areas around the eyes, nose, mouth, and jawline, among others. The technology leverages computer vision (CV) and machine learning (ML) algorithms to detect and follow these points as the face moves or changes expression. FPD is critical and has been studied intensively in recent years. The development of real-time FPD has been significantly influenced by progresses in several fields, like image processing and hardware advancements. The importance of this technology is evident across multiple applications, such as augmented reality and biometric authentication [1].

Artificial intelligence (AI) and deep learning (DL) have shown their utility and power in resolving numerous real-

Safa Teboulbi, Seifeddine Messaoud, Mohamed Ali Hajjaji, Abdellatif Mtibaa and Mohamed Atri have contributed equally to this work

✉ Safa Teboulbi
teboulbisafa@gmail.com

Seifeddine Messaoud
seifeddine.messaoud@fsm.rnu.tn

Mohamed Ali Hajjaji
mohamedali.hajjaji.issats@gmail.com

Abdellatif Mtibaa
abdellatif.mtibaa@enim.rnu.tn

Mohamed Atri
matri@kku.edu.sa

[1] Higher Institute of Computer Sciences and Mathematics of Monastir, University of Monastir, Monastir, Tunisia

[2] Faculty of Sciences of Monastir, University of Monastir, Monastir, Tunisia

[3] Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse, Sousse, Tunisia

[4] Systems Integration and Emerging Energies Laboratory (LR21ES14), National Engineering School of Sfax, University of Sfax, Sfax, Tunisia

[5] College of Computer Science, University of King Khalid, 61421 Abha, Saudi Arabia

world issues [2, 3]. Their major provocation is, not only, to stop the demand for direct programming, but also, to induce an intelligent system that can incontinently acclimatize to the recent situations. Among several DL algorithms, deep convolutional neural networks (DCNNs) are the state-of-the-art in CV problems [4]. Since AlexNet earned the 2012 ImageNet contest thanks to its outstanding recognition capacity, more and more enhanced DCNN architectures have appeared and been considerably used in numerous CV tasks such as image classification of object detection (OD), facial recognition (FR), semantic segmentation, independent driving [5, 6], and mainly FPD [1]. Thus, a great success has been achieved in multitudinous applications and in several fields [7–9].

The intention accuracies of DCNNs are more high than standard algorithms. DCNNs need enormous quantities of memory access and computational resources which represents a computational defiance for the central processing units (CPUs) and absorbs a huge quantum of power [4]. However, due to the large number of layers and the advanced calculation complexity, the DCNN's topologies are difficult to implement on embedded systems [10]. To address these issues, hardware accelerators, such as graphics processing unit (GPU), digital signal processor (DSP), application-specific integrated circuits (ASICs), and programmable gate array field (FPGA) have been used [4, 11]. For example, the first option for the neural networks (NN) process is GPU platforms, thanks to their high computational capacity and flexibility to develop frameworks. Otherwise, FPGA-based NN inference accelerators are getting not only a research topic but also a promising candidate [12]. They have gotten a lot of press in the fields of automotive driver assistance (ADAS) and data center acceleration. It can be affirmed that the optimized hardware designs-based FPGA represents the next solutions trend that can outperform GPU by enhancing the DCNNs performance [13]. FPGA is suitable to support the rapid elaboration of CNN models and improve their performance, due to its flexibility, capability to reconfigure at appropriate cost, low power consumption, its short time to market (TTM) and energy effectiveness [10]. In this particular way, several FPGA-based accelerator configurations have been proposed with software (SW)/ hardware (HW) optimization methods [13, 14]. Despite their advantages, the existing FPGA-based SoC designs face several limitations and challenges, like complexity in design and development, resource constraints, power consumption and cost [15].

Exploiting dynamic partial reconfiguration (DPR) in order to empower a large accelerated CNN on FPGA with a small size and alleviate long reconfiguration overhead is the main attention of this paper. In this regard, we focus on the demonstration of the capabilities of the proposed scheme by implementing a deep marker detection network, which is based on a CNN architecture. The goal of this video/image processing application is to determine key points in human faces to interpret facial expressions and gestures. Thus, we create a HW/ SW intellectual property (IP) cores for every convolutional (CNVL) layer employed in the target application. To ameliorate productiveness, the HW IP cores are accelerated using Xilinx Vivado high-level synthesis (VHLS) and then adjusted to the layer's characteristics. Furthermore, these accelerated layers will be exploited in the HW/SW design in which the DPR methodology is applied to improve the application performances. The main contribution of this paper are summarized as follows:

1. We propose a GPU-based software implementation for DCNN-based face point detection.
2. We propose a CNN-based acceleration and optimization method using Xilinx Vivado HLS tool.
3. We propose a DPR-based hybrid architecture to improve the proposed approach performances.

Validating our suggested design involves a multi-faceted approach, including simulation tools, benchmark dataset, performance metrics, and different case studies. We used Xilinx Vivado HLS (high-level synthesis), to design and optimize the hardware, which let the DCNN model run efficiently. The high-level programming language C++ is used. To train and test the DCNN model, the ImagetNet dataset is employed. To evaluate the efficiency of the proposed design, we first used three metrics (accuracy, precision and recall) for the GPU's implementation. After that, we compared the execution time in SW and HW. Also, we evaluated the resource utilization in terms of Block Ram (BRAM), Look-Up Tables (LUTs), Flip-Flops (FFs), and Digital Signal Processors (DSPs), in order to discuss the hardware cost of the FPGA in our CNN-based acceleration and optimization method. We proposed four scenarios to evaluate our proposed DPR-based hybrid architecture. We started with FPGA-SoC-based ARM implementation to the reconfiguration by IP cores selection (All IP cores are places in the PL side of the FPGA, only CNVL1 with CNVL3 in the PL and the others in PS side, and only CNVL2 with CNVL4 in the PL and the others in the PS.

The implementation of the F1 network on a GPU achieved high performance metrics with an accuracy of 89.01%, precision of 91.63%, and recall of 90.25%. Hardware acceleration using HLS significantly improved execution times for CNVL layers, with the second CNVL layer being accelerated by 45.01 times and convolution 1 by 41.8 times compared to software. The synergy between GPU power and HLS-based acceleration reduces computation time, enhancing neural network processes. Hardware acceleration through DPR was explored across four scenarios, with scenario 4 proving the most efficient, achieving a 2.3 times acceleration over software by optimizing core placement and utilizing co-design techniques.

The rest of the paper is divided into seven sections. Section 2 reviews related work based on DL models. The proposed work for facial point detection is presented precisely in Sect. 3. the GPU Implementation and training results are discussed in Sect. 4. Then, Sect. 5 provides the deep convolutional network acceleration for embedded facial point detection (EFPD). Yet, Sect. 6 presents the proposed HW/SW CNN design based on the DPR technique. Experimental results to evaluate the proposed approach performances are presented in Sect. 7. Finally, Sect. 8 concludes the paper and presents an overview of possible improvements.

2 Related work

One of the most useful DL approaches is CNNs architectures. Several improved CNN architectures have been widely used in CV tasks, such as autonomous driving, Hand gesture recognition (HGR), medical care, and FR. In this section, the state-of-the-art related to our context is clearly accentuated.

2.1 Implementation on multi-FPGA cluster

FPGAs offer several benefits including reduced development costs, shorter development cycles compared to ASICs, and lower power consumption compared to GPUs. However, it is worth noting that high-end FPGAs, which offer robust computing capabilities and extensive on-chip memory, often come with a higher price tag. Furthermore, despite the capabilities of high-end FPGAs, there are inherent limitations in terms of available resources within a single FPGA. To overcome this challenge, authors in [16] engineered a solution known as the PYNQ cluster. This innovative cluster comprises economical Zynq boards, referred to as M-KUBOS, designed to maximize resource utilization and performance.

The PYNQ open-source software platform is utilized to develop a PYNQ cluster intended for multi-access edge computing (MEC) services in 5 G mobile networks. Specifically, the ResNet-50 inference accelerator was implemented on this cluster for image recognition tasks in MEC applications. To optimize processing efficiency, the execution time of each ResNet-50 layer was estimated and distributed across multiple boards within the cluster. The implementation on 4 boards yielded impressive performance metrics, achieving 292 GOPS performance, 75.1 FPS throughput, and 7.81 GOPS/W power efficiency. Notably, this implementation surpassed CPU and GPU performance, achieving 17 times faster speed and 130 times more power efficiency compared to CPU, and 5.8 times more power efficiency compared to GPU.

2.2 Scalable acceleration of DNNs

The expansion in size and complexity of NNs in DL applications presents challenges for achieving high-performance implementations. Traditional solutions like GPUs and ASICs struggle to balance efficiency and adaptability. Authors in [17] suggested a SoC approach to accelerate DNNs. In this setup, an ARM processor oversees execution and delegates intensive computations to a specialized hardware accelerator. Experimental findings on a SoC development board demonstrate a 22.3× speedup compared to native implementations on a dual-core Cortex ARM A9 processor, using a 64×64 network architecture. A mathematical formula is introduced to estimate total execution time for varying architecture sizes, validated with a case study on Epileptic Seizure Recognition. Comparative analysis with state-of-the-art solutions assesses execution time, scalability, and clock frequency, validating the effectiveness of the proposed solution.

2.3 Cascaded multilayer perceptron neural networks

Hand gesture recognition (HGR) is increasingly becoming a popular means of interacting with human-computer interfaces (HCIs). In this context, authors in [18] proposed a hand gesture tracking system implemented on low-cost FPGAs using a multilayer perceptron and a supervised sequential learning algorithm. The system employs two neural networks: the first detects skin while the second identifies face and hand gestures. After training in Matlab, these networks are implemented on separate Xilinx Spartan XC3S1000 FPGAs, which are interconnected. The first network outputs a binary image identifying skin, which serves as input for the second network that distinguishes between hands, faces, and classifies hand gestures. The networks achieve a balance between accuracy and speed, operating at 208 fps with minimal memory use. The system, which can recognize five specific hand gestures with up to 97.3% accuracy before FPGA implementation and 96.33% post-implementation, outperforms other methods including those based on single-threshold skin color detection, without requiring backlight equalization for skin detection.

2.4 Hardware accelerator for multiple CNNs

Several accelerators have been proposed to cope with the increasing of CNN's computation complexity. In this context, authors in [6] proposed an FPGA-based CNN accelerator. The proposed design can support the acceleration of different networks. However, to do so, an automatic mapping flow is presented. The HLS is used to design the accelerator, which will be implemented on the Xilinx Zynq-7000 SoC. This SoC consists of two parts: the Processing System (PS) and the Programmable Logic (PL). The CNN accelerator depicts the PL

side. Three main blocs are the base of the PL: On-chip buffers, memory controller, and Processing Elements (PEs). A fixed point quantization strategy is adopted, in order to reduce the consumption of all resources. To evaluate the system, CNN model files of two networks, ZynqNet and SqueezeNet, are the inputs. These networks achieved both the acceleration on the design. A golden retriever is the input test image. Its corresponding label is 207, in the ImageNet dataset. The proposed accelerator is then compared with other designs. After the comparison, they note that the number of DSP and the throughput are confidently correlated. Consequently, the resources consumed by the design would automatically reduce the throughput. Despite the support of multiple networks of this accelerator, a functional redundancy occurs, especially for small networks like the ZynqNet. Therefore, the performance of SqueezeNet is higher than the one of ZynqNet. The performance density reached is 0.054 GOPS/DSP and the power efficiency is 5.24 GOPS/W when accelerating the SqueezeNet network.

2.5 FPGA for accelerated DNNs

FPGA is an up-to-date choice for CNN acceleration, favored for its low power consumption, and flexible design. Current research addresses the balance between FPGA's energy requirements and CNN performance, proposing solutions like the one in [4] which accelerates AlexNet's forward path, incorporating feature extraction and classification, on FPGA. The enhancement hinges on layer parallelism, pipelining, and on-chip memory usage. AlexNet is one of the well-known state-of-the-art in CNNs, distinguished by its top-1 and top-5 error rates, equal to 37.5% and 17.0% on the dataset. Acceleration begins with CNVL layers, relying on MAC operations performed by parallel engine (PE), composed of multipliers and accumulators. After comparing the execution time of the CNN on the FPGA with one of the few layers on MATLAB R2014a (CPU), they concluded that the FPGA is faster than the CPU. And after another comparison with different GPUs, GPU is faster than FPGA. Nevertheless, the FPGA is the most preferred thanks to its lower consumption, which is equal to 0.785 W, in the case of using Virtex7 FPGA. Finally, FPGA is an excellent platform for hardware implementations.

2.6 Depthwise separable convolution

Due to the large number of operations and parameters in standard NNs, it is difficult to deploy them in embedded devices. To dramatically reduce operations and parameters with limited loss of accuracy, some state-of-the-art CNNs have replaced standard convolution (SC) operations with depthwise separable convolution (DwSepCv) operations. Among these CNNs, we notice MobileNetV2, Xception, and ShuffleNet. In this context, authors in [10] proposed a scalable and high-performance accelerator named DPU (deep learning processing unit) for MobileNets which is based on FPGA. MobileNets are built on a well-organized structure that employ DwSepCv to generate light DNNs. The DwSepCv is a shape of the factorized SC. For effect, in DwSepCv, the SC is split into two types of convolution, which are DwCv and pointwise convolution (PwCv). Noting that the targeted platforms used for implementing the DL accelerator are the Xilinx XCZU2EG also the XCZU9EG MPSoC. Three implementations are developed in this work. The first one is the acceleration with channel augmentation (CA) in DPU. The speedup of the first layer is 1.31× on ZU9, but on ZU2 is 1.91×. The efficiency is improved by CA, from 3/12 to 9/12 and from 3/16 to 9/16, respectively, on ZU2 and ZU9. Hence, we can say that the acceleration of CA is more crucial on ZU2 board than on ZU9 board. The second one is the classification on both ZU2 and ZU9. Therefore, the accelerator achieved a performance 2.13 times much faster than the one on ZU3, despite the supplementary logic resources on the ZU3. On ZU9, the performance of the accelerator is 3.04 times rapid than the best. The third and the last one is the implementation of DPU in MobileNet V1 + SSD on ZU2 and ZU9 MPSoC, for OD. The performance of DPU in OD is compared just with CPU. Therefore, the accelerator accomplishes 8.4× and 33.6× speedup on ZU2 and ZU9, respectively when compared to the CPU. After discussing implementation results, we can say that MobileNets with little model size possess a competitive accuracy, not only on classification but also on OD. This important power enables MobileNets to practice CV tasks. As a conclusion of this work, the DPU accelerator can be exploited to numerous devices flexibly, with distinct configurations for the purpose of balancing HW resources, as well as the computational performance.

2.7 CNN customization

CNVL, Pool, and FC are among the various types of CNNs layers. The performance of CNNs is largely affected by the CNVL layer, which traditionally relies on spatial algorithms for convolutions. Recent advancements include the use of FFT and Winograd algorithms to reduce computational complexity, although they come with their own limitations regarding parameter compatibility. Despite the diversity in layer types, parameters, and algorithms enhancing CNNs, most FPGA-based accelerators use static designs, leading to underutilization of resources and inefficiencies. To counter these issues, the authors in [19] proposed an FPGA-based framework called Fune, in order to address computation inefficiencies cited previously. Fune offers a novel method for highly personalized and efficient optimization of CNNs on FPGAs by utilizing the reconfigurability of FPGAs which allows the customization of the HW resources for various

layers and parameters. It employs a sophisticated tuning algorithm consisting of dynamic reconfiguration and optimal configuration search strategies. Fune generates targeted bitstreams for each configuration, employing high-level programming to ensure efficient compilation through Vivado tool-chains. To illustrate the effectiveness of Fune, four CNNs are employed as benchmarks (AlexNet, ResNet, DispNet, and GoogleNet) using two FPGA SoCs (Xilinx Zynq ZC706 and ZCU102), demonstrating significant improvements over traditional spatial and Winograd designs. These tests, performed at 166 MHz with 16-bit fixed-point data on the Vivado SDx platform, highlight Fune's potential in enhancing FPGA-based CNN performance through adaptive and efficient design customization. For instance, when comparing the speedup of the four tuned designs produced by Fune with only spatial designs on the two FPGA-SoCs, a 2.17× speedup is reached at the moment that the batch size is 64, for the AlexNet network. For GoogleNet and ResNet which are in-between cases, the reconfiguration is not completed while the batch size is small because of the expensive cost of the overhead. As the batch size remains growing, a tuned strategy accompanied by reconfigurations is obtained and persists unchanged. This indicates that even without reconfiguration, Fune outperforms uniform designs by mitigating computational inefficiencies. For each reconfiguration, as much as utilized resources by Fune, in order to improve parallelism, the power results of the different designs are very close. Concerning the overall energy consumption, an average of 41.8% and of 35.6% of energy is saved by the designs produced by Fune when comparing them sequentially, with spatial and Winograd designs, on the ZC706 chip. But on ZCU102, the saved energy is 40.7% and 33.9%, respectively.

2.8 Bit-width partitioning

For each CNN layer, the current majority of FPGA-based techniques used bit-width selection, which leads to extremely low resource utilization efficiency and performance improvement issues. In this context, authors in [8] proposed a new approach using bit-width dividing of FPGA DSP resources in order to raise the CNN accelerator efficiency. The DSP resources on the FPGA-SoC are partitioned into multiple distinct bit-width parts. Then, in the CNN accelerator map, these DSP parts are used to build numerous diverse bit-width CPs. In a parallel process, these CPs exploit the CNN layers. As a consequence, it is justly high-efficiency to operate images for the CNN accelerator.

An optimization approach is also used in this work for detecting the ideal allocation plan for the DSP resources, between diverse processors in order to acquire the best CNN accelerator design. By conducting different well-known CNNs and using VHLS 2015.4.2 found on Xilinx Virtex-7 485T FPGA MPSoC, the validation results are presented and then compared with the state-of-the-art approaches. In fact, the novel design approach reached the highest performance improvement; it is 5.48× for AlexNet, 5.88× for VGG-16, and 7.25× for VGG-11.

2.9 Novelty

The novel contribution of our work lies in its proposition of a DPR and hybrid architecture tailored for the acceleration of DCNNs, particularly focusing on FPD applications. This innovation diverges from traditional approaches by merging the computational power and flexibility of GPU-based software implementations with the energy efficiency, low latency, and high reconfigurability of FPGA-based SoC designs. Specifically, the use of the Xilinx Vivado HLS tool for CNN optimization signifies an advancement toward simplifying the design process, making it more accessible and efficient. Furthermore, the introduction of DPR into the FPGA architecture allows for real-time reconfiguration of the system without halting its operation, offering unparalleled flexibility and performance enhancement. By validating this approach across four distinct scenarios, the paper not only showcases the practical viability of its proposed architecture but also highlights its superiority in terms of reconfigurability, execution time, hardware cost, and power consumption when compared to existing solutions in the domain of AI-driven computer vision.

3 Proposed work

The proposed work, as depicted in supplementary material of figure 1, consists of three main parts. The first part of this work aiming at developing a software based GPU framework for the face point detection. In this phase the adopted model, the used dataset, the training and the testing dataset, and the tuned hyperparameters are well introduced (for more details see Sects. 3.1 and 3.2). The second part consists in accelerating the CNN layers which require high computational capacities and cause an execution delay which does not meet the requirements of real-time embedded vision applications. To do so, the Vivado HLS of xilinx is used to synthesize and optimize the layer's functions (for more details, see Sect. 5). On the other hand of this part, the HW/SW architecture is designed to implement the proposed facial point detection as a fusion between software and hardware parts. In order to provide a powerful HW/SW design with low latency and power, and high throughput with low hardware cost, the DPR technique is used in the proposed design (for more details see Sect. 6.1). In the last part, the hardware files are generated, and the customized OS is created to implement the proposed co-design on the PYNQ-Z1 MPSoC.

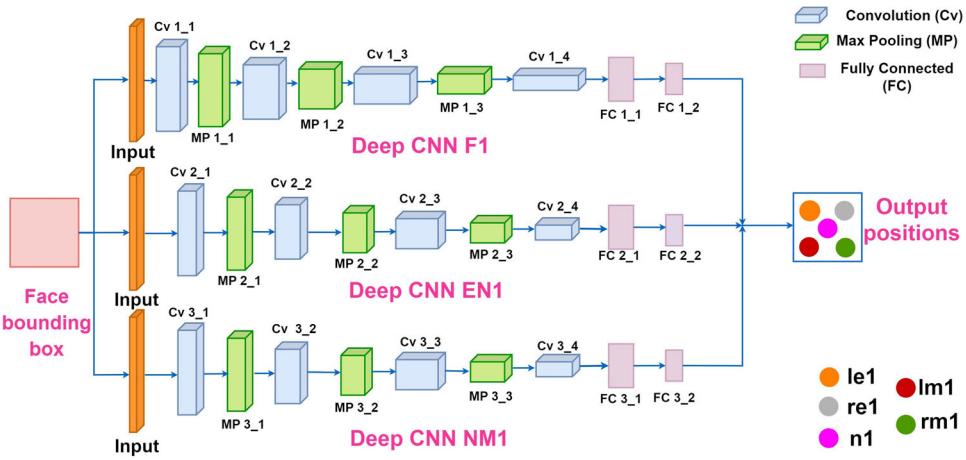


Fig. 1 Structure of cascaded deep CNNs

3.1 Deep convolutional network for facial point detection

The recognition of facial keypoints (FKPs) is an important and challenging problem in computer vision and machine learning fields. Predict the coordinates of the FKP, such as the nose tip of a particular face, the center of the eyes, etc. The problem is to predict (x,y) coordinates in the image pixel space of the FKP for a given face image. Its applications are tracking human faces in images and videos, detecting deformed facial signs for medical diagnosis, analyzing facial expressions, facial recognition, and more. Facial features vary from person to person, and even for an individual person. This problem becomes more difficult when capturing images of faces under different lighting conditions, extreme poses, viewing angles, occlusions, expressions, and flash. Generally, there are two categories for existing approaches. The first category is for classifying search windows. The second is intended to directly predict the positions of key points (KP), which is more methodical than the first, because it does not require scanning [1].

To solve all these problems, authors, In [20], proposed a multilevel regression method using a deep cascade CNN to predict facial keypoints (FKPs), crucial for facial detection and analysis. Unlike traditional approaches, which train independently for each KP or predict KPs using regressors, their method simultaneously predicts accurate positions for all FKPs at the first level using CNN cascade. These FKPs include LE (left eye center), RE (right eye center), N (tip of the nose), LM (left corner of the mouth), and RM (right corner of the mouth). Level one networks are complex with four CNVL layers, extracting local features using lower layers' neurons. Unique aspects include using tanh activation for all neurons and local weight sharing in CNVL layers, dividing input into four sections to convolve with region-dependent kernel filters. This approach aims to capture unique characteristics of eyes and mouth separately. Pool layers' biases match CNVL layers', enhancing feature extraction.

3.2 Cascaded CNN architecture for robust facial keypoint detection

Our approach utilizes the entire face as input for CNNs, maximizing texture context while filtering high-level features. Despite potential ambiguity, CNNs predict keypoints simultaneously, implicitly encoding their limitations. Different CNNs are designed for each cascade step, with shallower structures for low-level tasks. Multiple CNNs are merged at each level to enhance reliability. Factors influencing network structures for effective face point detection are identified empirically. The cascaded CNNs detect five facial points: LE, RE, N, LM, and RM, facilitating coarse-to-fine prediction across three levels.

At level one, three deep CNNs are employed, F1, EN1 and NM1, as mentioned in Fig. 1; their input regions cover, respectively, the whole face, eye and nose, and nose and mouth. Each one of these networks predicts simultaneously various FKPs. And the predictions of several networks are averaged in order to decrease the variance, for every facial point. In F1 deep structure, it exists four CNVL layers ensued by one max Pool layer and two layers of FC. The same deep structure is taken by EN1 and NM1, but with distinct sizes at all layers since their input regions' sizes are different. At levels two and three, the networks take the local fixes centered on the predicted positions of the FKPs from the previous levels as input, and they are rightly allowed to make small changes to the previous predictions. Patch sizes and search ranges continue to decrease along the waterfall. The predictions of the last two levels are strictly compacted because sometimes the local appearance is unreliable and ambiguous. Moreover, at the last two levels, the predicted position of each point is specified by the average of two networks with distinct patch sizes. At the time when the networks aim to strongly predict

the positions of the KPs with large errors at the first level, the networks at the other two levels are represented to achieve high accuracy. In addition, all networks at the last two levels share a common lightweight structure since their loads are low level.

We recognize different effective ways to combine many CNNs. Multilevel regression is the first level. In the latter, the unique knowledge for networks is the delimitation of faces. Due to the instability of face detectors and large pose variations, the relative position of FKPs with respect to the bounding box can differ over a wide range. Therefore, at the first level, the input regions of the networks must be wide to cover many possible predictions. However, the main cause of inaccuracy is the large input region because the included unimportant areas can degrade the final output of the grating. Thus, the detection of the second level can be done inside a small region.

The proposed cascade CNN with n levels is represented by Eq. 1, where l_i denotes the predictions at level i. Noting that at the first level, those predictions are perfect positions, while at the following levels, the predictions are adjustments. Noting that $I(h, w)$ denote the input layer which is 2D in the case of not using the color information, where h is the height and w is the width, of the input region. $CR(s, n, p, q)$ indicate the convolution layer, if the rectification of the absolute value is employed, otherwise is considered as $C(s, n, p, q)$. The length of the stride of the square convolution kernels is denoted by s, the maps number in the CNVL layer is denoted by n, and the weight sharing parameters are presented by p and q. Every map in the convolution layer is uniformly split into p by q regions and in each region, the weights are shared locally.

$$y_{i,j}^{(t)} = \tanh\left(\sum_{r=0}^{m-1}\sum_{k=0}^{s-1}\sum_{l=0}^{s-1} x_{i+k,j+l}^{(r)} \times w_{k,l}^{(r,u,v,t)} + b^{(u,v,t)}\right) \tag{1}$$

where x and y are respectively the outputs of the previous and the current layers. The weight is denoted by w and the bias is symbolized by b. The m maps in the preceding layer are associated with m s by s kernels. The resulting maps, along with a bias, passed the tanh nonlinearity after being accumulated, creating one of the n map in the CNVL layer. For various output maps and distinctive regions in the maps, the kernels set and the bias are distinct. $CR(s, n, p, q)$ is analogous but has a supplementary operation after tanh. $P(s)$ represents the Pool layer and the length of the side of the square Pool regions is designed by s. Max pooling (MP) is utilized and the Pool regions are not imbricated. Pool results are firstly multiplied with a gain coefficient (g), secondly shifted by a bias (b), and thirdly followed by a tanh nonlinearity. The two coefficients: gain and bias, are shared identically at the preceding CNVL layer. $P(s)$ is defined in Eq. 2, in which $F(n)$ denote the FC layer that is denoted by Eq. 3).

$$y_{i,j}^{(t)} = \tanh\left(g^{(u,v,t)} \times \max_{0\le k,l<s}\left\{x_{i\times s+k,j\times s+l}^{(t)}\right\} + b^{(u,v,t)}\right) \tag{2}$$

$$y_j = \tanh\left(\sum_{i=0}^{m-1} x_i \times w_{ij} + b_j\right), for: j = 0, \ldots, n-1, \tag{3}$$

4 GPU implementation results

In this section, we present the training results of the adopted CNN topology. The overall experiments were implemented on an Intel Core TM i7-3770 @3.4GHz CPU and 16GB RAM. We also use the NVIDIA GeForce RTX2070 GPU to improve the speed of the proposed trained deep model. The hyperparameters used in our experiments are summarized as follows: the learning rate which is set to 0.0001, the batch size of 32 and the number of training epochs is of 500, while the Adam optimizer is used to update and tune the network's weights. The training and validation accuracy (Eq. 4) and the training and validation loss [21], together with the evaluation parameters as well as the accuracy, precision and recall, are used to evaluate the performance and to show how efficiently the model "learns" of the topology adopted in our work. The idea of this work is to train each level of the adopted cascade network only on the same dataset and to select the optimal model which will be used in the second contribution. Supplementary material of figure 2 shows the model accuracies and losses achieved during the learning process of level 1, level 2 and level 3, respectively.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{(\text{TP} + \text{FP}) + (\text{TN} + \text{FN})} \tag{4}$$

The level 1 network results shown in the left subfigure prove that the CNN topology (level 1) achieves an accuracy of 89.01% without any overfitting issues. In addition, the precision curves of the train and the validation are in the same format. Regarding the loss of training and validation, the curves presented (loss of train and validation) prove that the first level of the network reaches a good convergence. For the EN1 level, the train and validation accuracy, in the middle figure, reached a maximum accuracy of about 86.32% while the train accuracy curve and the validation accuracy curve exhibit a small deviation which can then affect the correct prediction. By comparing the results of the EN1 level to the first, we conclude that the first level could be the best CNN network. The last figure (right) provides the train and validation accuracy that achieved 85.42% accuracy with a perturbation curve that decreases model stability. In this regard, supplementary material of table 1 presents the classification performances of the three levels of the adopted

Table 1 Classification performances of the proposed CNN

Model	Accuracy (%)	Precision (%)	Recall (%)
Level F1	89.01	91.63	90.25
Level EN1	86.32	89.86	90.03
Level MN1	85.42	89.08	91.71

CNN network. Accuracy (Eq. 5), also called Positive Predictive Value (PPV), is a sufficient measure for determination, while the cost of false positives is high. The recall (Eq. 6) is the representation of a model used to choose the best model especially when there is a high cost associated with a false negative. TP stands for true positives, TN stands for true negatives, FP stands for false positives, and FN stands for false negatives.

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \tag{6}$$

in this respect, as shown in Table 1, the F1 level network obtained the highest accuracy, which is about 89.01%, and a precision of about 91.63% compared to the EN1 level which achieved an accuracy of 89.86% and a recall of about 90.03%, while the MN1 level network achieved an accuracy of 85.42%, a precision of 89.08% and a recall of 91.71%. Based on these results, the CNN-based F1 level network is selected among the other levels to be accelerated and implemented on an embedded FPGA-SoC vision system exploiting the technique of dynamic reconfiguration.

5 Acceleration of deep convolutional network for embedded facial point detection

The idea is to accelerate the selected CNN topology and then design a HW/SW co-design which will be implemented on the PYNQ-Z1 FPGA-SoC. To do so, the weights and bias values are extracted using a created parser from the trained model. Therefore, two predicted file after parser application, the first one store the weight and bias while the second file summarize the whole network description and interconnection. In this context, the Xilinx Vivado HLS is used in order to produce layer-specific IP cores to design hardware implementations. The retrieved weights and bias are used in the HLS testbench to test the developed IP cores. The overall HW/SW architecture is designed with the Xilinx Vivado IPI, in which the high-definition multimedia interface (HDMI) is used to for video capture. This section highlights the CNN acceleration flow, the dynamic partial reconfiguration technique, and the HW/SW co-design.

5.1 HLS-based CNN acceleration

The PYNQ-Z1 SoC platform that will be used is a Xilinx Zynq-7000 ZC7020 that consists of both processing system (PS) and programmable logic (PL). The PS part is a dual-core ARM A9 hard-core processor with several peripherals, while the PL part consisted of an FPGA. Two HDMI ports are also included in the PYNQ-Z1 SoC. These HDMI ports are used in order to receive the input image and then produce the output image, which includes facial landmarks. The signal pipeline of the HDMI is implemented within the PL HDMI subsystem. As a frame buffer, the IP Video direct memory access (VDMA) core is used. We separate the two subsystems using independent frame buffers owing to the considered the predicted mismatch in clock cycles between CNN pipelines and HDMI.

The data that is fed into the CNN hardware IP cores is connected through a DMA IP core. Thus, unlike VDMA, transmission must be triggered for each frame. In this context, the input data are preprocessed in the PS side. Faces are detected, designated with a bounding box, and cropped from the picture in the first phase. Therefore, each image, which will be resized later, includes a face detected and then converted to gray-scale format. Then, these images are converted to the standard array format. As the last step in preprocessing, subtracting the mean of the pixel values and dividing by the standard deviation of the pixel values are applied to all pixels in the image before it is passed to the CNN. A detailed layer's description of the F1 CNN network is shown in supplementary material of table 1. Each CNVL layer is characterized by its parameters set. Three distinct kernel sizes 2×2, 3×3, and 4×4, should is considered. In addition, various channel dimensions and the parameters' number which will be exploited to configure CNN layers are considered for this work. The main aim is to configure different parameters according to the characteristics of each CNVL layer to provide the best performance while taking into account the constraints of the hardware resources. As illustrated in supplementary material of table 1, the second convolution (CNVL2) owns the highest computations number while the fourth convolution (CNVL4) possesses the minimum computation to communication ratio. Nevertheless, these adjustments must be realized in a time-saving manner. Thus, we select a high-level design entry accompanied by a guide implementation, from which we can deduce any HW convolution IP core.

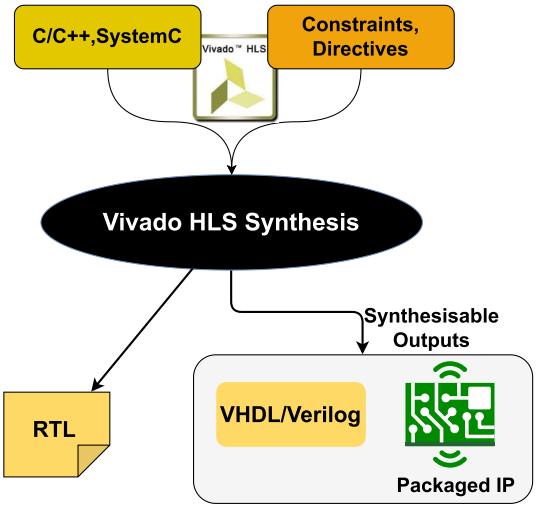


Fig. 2 HLS process

5.2 HLS tools

Recently, FPGAs have become a valuable heterogeneous computing component. Register transfer level (RTL) design flows require hardware knowledge from designers. In this vein, Vivado high-level synthesis (HLS) is the most widely used tool, which converts a SystemC or C/C++ source into an RTL implementation [22]. The ZYNQ board is adopted to synthesize and implement this design. Figure 2 depicts the overall design flow, which includes the main procedures and outputs with additional phases as particular verification components. In terms of HLS process inputs, this is a software function that was tested, designed and validated using a C-based testbench. This will entail a benchmark against which the outputs produced by the function intended for synthesis will be tested. This might be in the form of a preset set of output values or it can be built right into the testbench. Following the completion of the HLS process, output files containing design files for the desired RTL language are generated [23]. The final step is to package the RTL output after the design has been validated and the implementation has been successfully passed. This can be done directly using the RTL files generated by the HLS process; however, it may be easier to package IP using Vivado HLS capabilities. Because of the packaging of Vivado HLS output, HLS designs may be readily integrated with other Xilinx tools, such as Vivado IP Integrator [24].

5.3 CNN acceleration

The convolution (CNVL), the pooling (Pool) and the fully connected (FC) layers are three different types of CNN layers. The parameter sharing and local connections' concept are not exploited in the FC layers, contrary to the CNVL layers. Due to the large number of parameters required for matrix multiplication, these layers have a very low computation/communication ratio. Additionally, current CNN architectures, such as GoogleNet, have effectively replaced the FC layers with average Pool layers. Therefore, we suggest that the FC layers be implemented on the PS of the SoC first. The CNVL, activation function, and Pool layers are all implemented as separate HW IP cores on the PL side. Different IP cores may be beneficial for utilizing the reconfigurable region in a powerful way depending on the reconfiguration flow and the sequential structure of the CNN inference path. On the other hand, high-level design input is advantageous for creating hardware implementations that accelerate and leverage the functionality of each layer independently. HLS is used to quickly generate HW IP cores. Xilinx VHLS generates hardware RTL descriptions from a high level of algorithm-level abstraction, such as C/C++. This is an automated process, but also pragmas and directives to influence the planning and allocation process can be used.

The AXI is used to create the communication infrastructure. The data exchange which is based on Handshake is supported by AXI4-Stream between a receiver and a sender. There is no addressing overhead for the reason that it is not memory-mapped. AXI4-Lite is a memory-mapped protocol that works similarly to AXI4. The key distinction between the two types is AXI4's support for burst transmissions. The PS utilizes an AXI4-Lite interface to monitor and also control each layer. A high data throughput is required because of the CNNs data-driven flow. Hence, for input, output, and intermediate data transferring, we employ AXI4-Stream-based interfaces. CNVL layer data transfer is dependent on payload data and another processing parameters. As a consequence, to transfer the weight and the bias parameters, we use AXI4 interfaces (memory-mapped) with a regulated burst size. The data throughput is lower than the layer's input data owing to the CNVL layers' weight sharing principle. Therefore, feature maps are firstly fed forward in width/height order first, and then in-depth order. The Pool layer might then be applied in a cascade without buffering. Each CNVL layer employs an input line buffer and an output buffer to regulate intermediate data in the CNVL layer. The convolution result is then transferred to the output stream. Figure 3 shows the basic principle of the line buffer. The AXI4-Stream feeds on-chip memory with sequential incoming data, which is then shifted left as new data arrives until it reaches the end of the line buffer. Information is deleted from memory once it reaches the upper left storage. After that, the technique can be applied by convolving a nucleus into a single region. Therefore, if the required resources are accessible and the row buffer implementation allows nine simultaneous storage accesses, nine multiplications will be processed in parallel. The row buffer size and partitioning must be adjusted if mul-

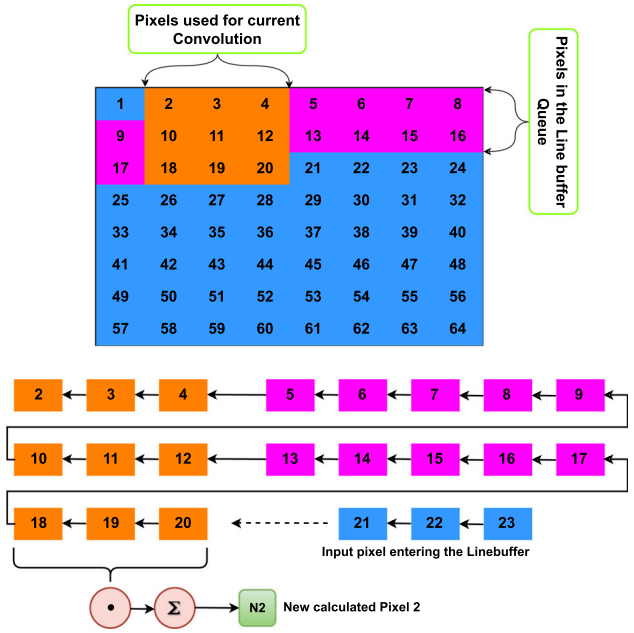


Fig. 3 Line buffer technique

tiple convolutions are run in parallel. By increasing the row buffer size by one element, sixteen simultaneous multiplications can be performed. By default, due to restricted access, VHLS implements this row buffer as a dual-ported LUTRAM or BRAM instance, limiting additional parallelism using pragma array.

In CNNs topology, the CNVL, which is given by Algorithm. 1, is the most computationally intensive layer. Cores are convolved with specific properties, along with paddingand stride, across the full height, width and entry depth. HLS provides many pragmas in order to tune resource allocation and scheduling in the convolution process. Unwinding and pipeline are among the most important pragmas for optimizing loop execution scheduling. As a result, the loop can be scheduled, allowing more operations to be performed in parallel or a method of pipelining. The line buffer size, loop scheduling, allocation operation, and partitioning consider a challenging task due to the constrained resources provided by the reconfigurable FPGA area. Considering resource constraints, the goal of this step is to provide the best compromise between resource consumption and performance of each layer. We suggest piping Loop4 with rewind quality for each convolution IP core due to limited on-chip resources. Therefore, Loops 5 and 6, in line 6 and line 7 respectively, unwind automatically. The pipeline technique strengthens the scheduler to share resources and reduce latency. The number of parallel operations that can be executed in parallel within this structure is based on the kernel size. Accordingly, for Loop 4 (line 5 in Algorithm), we used an additional unroll pragma with a particular factor depending on the core size. Allocation pragmas are further used to allow the scheduler to take resource restrictions into account. However, resource sharing has significant drawbacks in terms of routing, scheduling, and placement. The maximum frequency increases due to the recurring use of the same HW components. In addition, when BRAM resources are concerned, this issue becomes particularly critical. To facilitate this change, we are adding input/output (I/O) and registers for each BRAM instance and improving the clock uncertainty to ensure each IP core stays in synchronization with 10 ns clock period. We employ floating point quantization operations to improve performance and decrease resource usage. Thus, each operation employs a 32-bit bit-width data type, which includes eight full bit-width lengths. The impact of this quantization on accuracy is less than 1%, so it is not considered as a big deal. Due to the characteristic of the regression task, where the loss of precision is much more visible than in classification tasks, we suggest not reducing the precision of parameters and inputs. Therefore, this quantization improves the performance of each IP core convolution by reducing the hardware cost.

Algorithm 1 Convolution layer

image_size (x, y), image_depth, stride(s), Filter_x, Filter_y, Filter_size_x, Filter_size_y, Filter_Number, Weights CNV Initialize: CNV=0

```
for Filter=1:Filter_Number do
    for x=1:image_size(x) do
        for y=1:image_size(y) do
            for z=1:image_depth do
                for Filter_x=1:Filter_size_x do
                    for Filter_y=1:Filter_size_y do
x_n = Image_x + Filter_x
y_n = Image_y + Filter_y
CNV = Weights(Filter_x, Filter_y, image_depth, Filter) *
input(x_n, y_n, image_depth)
End For

End For
End For
End For
End For
End For
```

Figure 4 shows the optimization-based HLS acceleration results for all CNVL layers. Because of the huge computations number provided, CNVL1, CNVL2, and CNVL3 need the highest time to calculate all dot products, which is about 243.1 ms in SW to 5.4 ms in HW, 59 ms in SW to 2.6 ms in HW, and 104.5 ms in SW to 2.5 ms in HW, respectively. This prove that these layers reached the acceleration, after the HLS process, by speeding up the software execution time by 22.96 times for CNVL1, about 45.1 times for CNVL2, about 43.54 times speeding up time for CNVL3, and about 24.48 times in CNVL4, as presented by Fig. 4c. This increase the hardware cost required by these layers, given the pipelining introduced, to accomplish all affected operations in the desired time. As shown in Fig. 4a, the CNVL1 IP occupies about 29% of Digital Signal Processor Instances (DSP48E), 24% of Block RAM (BRAM), 24% Flip-Flops (FF), and about 34% of Look-up tables (LUT). In addition, CNVL2 require about 20% of DSP, 21% of BRAM, 17% FF, and about 31% of LUT. At the same time, CNVL3 needs about 20% of DSP, 21% of BRAM, 14% FF, and about 28% of LUT. Due to the minimum operation number, as denoted in Fig. 4d, CVNL4 presents the minimum required hardware resources which is about 7% of DSP, about 24% of BRAM, 9% of FF, and about 18% of LUT. Thus, all CNVL IPs work at 100MHz frequency. As an additional results, Fig. 4b presents an approximation about the latency of the CVNL IP's which shows that the maximum latencies are scored by CNVL3 and CNVL4 that translates the maximum attributed MMAC number.

6 Proposed HW/SW CNN design based on dynamic partial reconfiguration technique

6.1 Dynamic partial reconfigurable-based block design

Dynamic partial reconfiguration (DPR) allows one part of an FPGA fabric to be reconfigured without affecting the remainder of the fabric's operations. To employ an FPGA as a dynamically shared computing resource, an FPGA structure may be partitioned and managed into several DPR partitions that can be independently reconfigured at runtime with various application functional units[25]. Therefore, considering the restricted reconfigurable resources, we design a reconfigurable block with a 28% of FF, 30% of BRAM, 40% of LUT, and 35% DSP. The HDMI infrastructure uses the leftover PL logic. Because of the enormous number of computations needed, CNVL2 takes the longest time to calculate all dot products. Distinct convolution layer communication-to-calculation ratios establish different enhancement criteria, preventing comparing performance at the same level. For the pooling layers, a line buffer is employed to buffer incoming data in addition to the hardware implementation of the convolution layers. The row buffer is divided into sub-arrays based on the input dimensions and kernel size property of the pooling layer, with the number of sub-arrays equal to the kernel height. The highest value may be computed in parallel in each of these sub-networks. The output value is the maximum of the results of all the pipelined sub-arrays. The number of line buffer shift operations is determined by the grouping

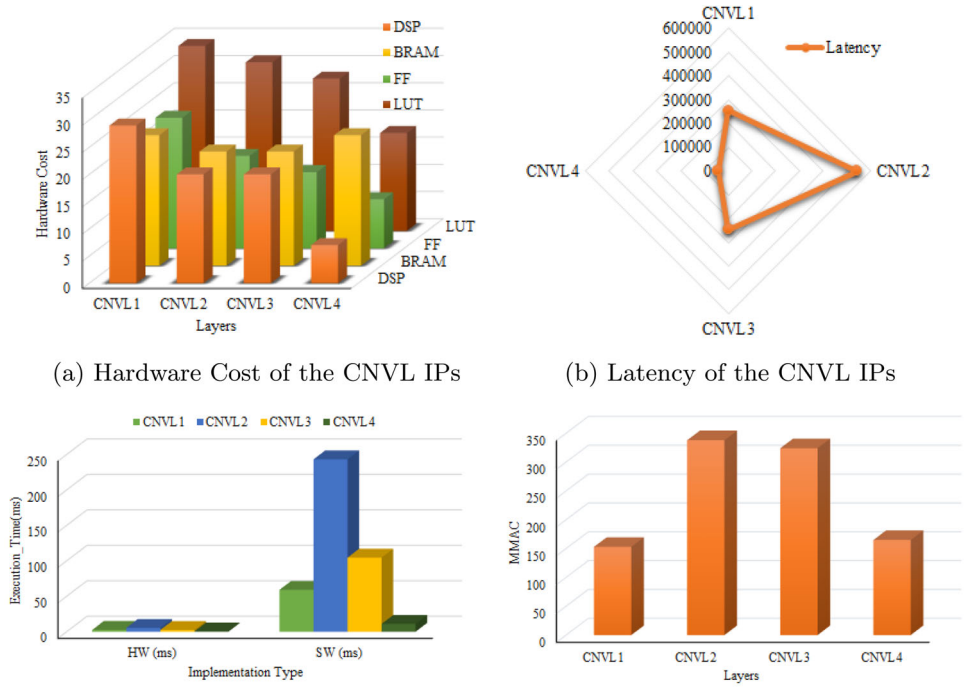


Fig. 4 HLS IP core optimization results of the convolutional layer acceleration

(a) Hardware Cost of the CNVL IPs

(b) Latency of the CNVL IPs

(c) Execution time of the CNVL IPs in HW and in SW

(d) Mega Multiply-Accumulate Operation per second

layer's stride property. Unlike hardware IP core convolution and pooling, the IP core enable feature works with unbuffered data streams. The max function based floating point technique for pooling and ReLu cores are implemented using a function instance given by the Vivado HLS math library that has been tuned for hardware implementations.

As already demonstrated, the CNN layers' HW implementations need more resources than the target device can provide. Furthermore, the CNN accelerator and the HDMI system share the same resources. Hence, we used DPR to partition FPGA resources between various CNN layers. It takes a lot of time to reconfigure an FPGA. Even with DPR, which reduces the timing overhead by reducing the reconfigured logic amount, the reconfiguration time remains a significant constraint in addition to the limited reconfigurable area. Therefore, the RP's size must be chosen taking into consideration all HW IP size. Figure 5 depicts RP's size and placement for our CNN framework. Because of the CNVL layers' size, we designed a single RP block to be used partially and dynamically by all CNVL IPs.

Figure 6 compares the contained resources with the resources used by the reconfigurable CNVL IPs. Figure 6a provides clarification on the percentage of reconfigurable hardware resources versus overall FPGA resources, while Fig. 6b gives the HW resources occupied by each CNVL IP compared with the reconfigurable HW resources. Layers are exchanged in and out of the RP according to the data flow within the CNN to provide HW acceleration of numerous layers. This procedure is carried out without touching the HDMI subsystem's remaining logic. The location of the RP is split into two horizontal clock parts. Due to timing limitations, we discovered various problems when using the RP above two or more vertical clock regions during our evaluation. Another limitation is the static logic's very high resource utilization, which is principally determined by the HDMI subsystem, as well as the utilized I/O pins. The overall clock sections, as represented by the right margin on the RP's side in Fig. 5, were not accessible. The purpose for this is that the HDMI subsystem's I/O pins are on the correct side. The reconfigured programmable logic reset is another issue that arises as a reconfiguration result. To avoid this, we used a manual technique to reset the IP cores after the reconfiguration. In this study, a simple and general purpose input/output (GPIO) IP is employed with an active low reset feature. The overall size of the bitstream is around 4.0 MB, which includes the static design and the partial bitstream size which is around 2.1 MB.

6.2 Reconfigurable HW/SW CNN design for facial point detection

A Xilinx Zynq-7000 ZC7020 is used on the PYNQ-Z1 board. It is built on a dual-core ARM A9 hard-core CPU and several peripherals such as a PS and a PL. Furthermore, the PYNQ-Z1 board includes two HDMI connectors. Thus, we leveraged these components in our partially reconfigurable

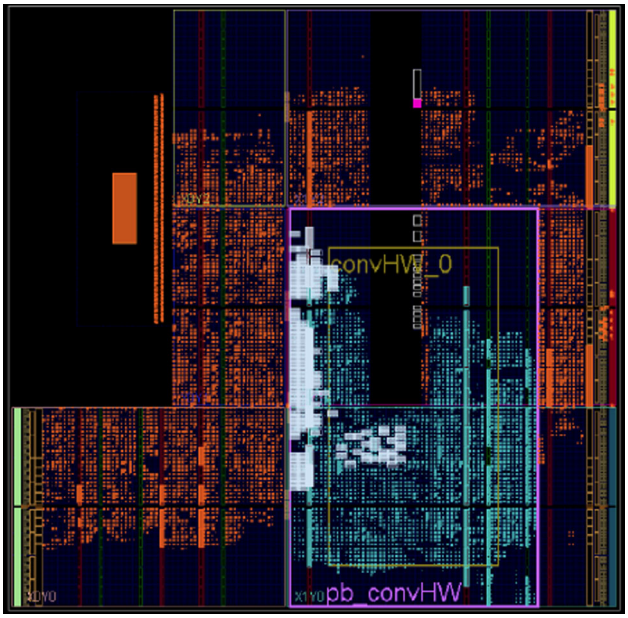


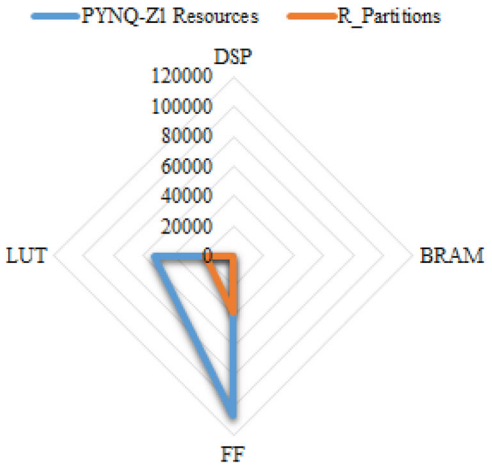
Fig. 5 Reconfigurable area (blue), static logic placement (orange), and the fixed I/O ports (white) (color figure online)

HW/SW design, as shown in supplementary material of figure 3. We used the HDMI ports to receive the input video and output the facial point detection results. The HDMI pipeline is operated and implemented inside the PL. The VDMA IP core is used as a frame buffer between the HDMI IP and the CNN IP, to avoid the clock cycle difference. Therefore, video input data is first stored in DMA before being transmitted to CNN IP. The PS controls and coordinates the overall system. PS begins by preprocessing the input data by detecting the face, drawing the bounding box and resizing it to 39×39 pixels. Next, the mean of the pixel values is subtracted and divided by the standard deviation of the pixel values, which is applied to all pixels in the image. After that, the data is stored in the necessary memory space with a CNN-DMA physical address. Subsequently, the configuration of the core IP is performed and the layer's weight and bias parameters are extracted from the file saved during the learning process. These parameters are subsequently transferred to the IP core and the DMA-memory, where processing may begin. The IP core's simulcast output is fed back to the memory DMA and written to memory DDR. As the data transmission is completed, the IP core is reconfigured. We employ the processor configuration access port (PCAP) interface and the driver to apply DPR. As a result, we used software to reconfigure the PL.

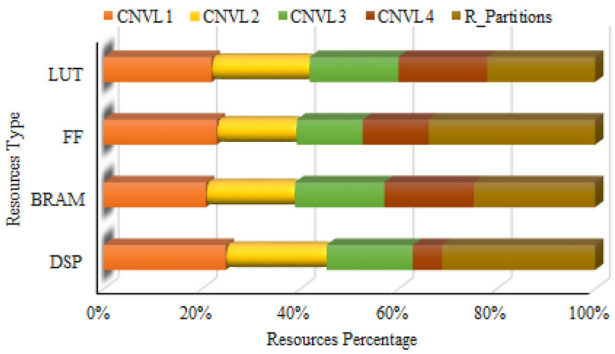
7 Results and discussion

7.1 Proposed work results

After all bitstreams have been generated, the overall embedded vision system for facial point detection must be managed and controlled via the PS. Our customized Overlay allocates the input/output memory areas which is executed on PS part of the PYNQ-Z1. The xlnk driver is used by the customized OS to reserve memory that is mapped to physical space to provide communication with the PL, containing frame buffers, memories parameters, and the IP cores communication's drivers. The first strategy in the experiments is to implement the CNN system on the PYNQ-Z1 ARM processor, as seen in Fig. 7a. Thus, the CNN layers are implemented on the ARM while the only the HDMI system is implemented on the PL FPGA. To assess this scheme, we evaluate the running time of the ARM-based CNN in processing ten images, as shown in Fig. 7b. The mean run time of the ARM-based

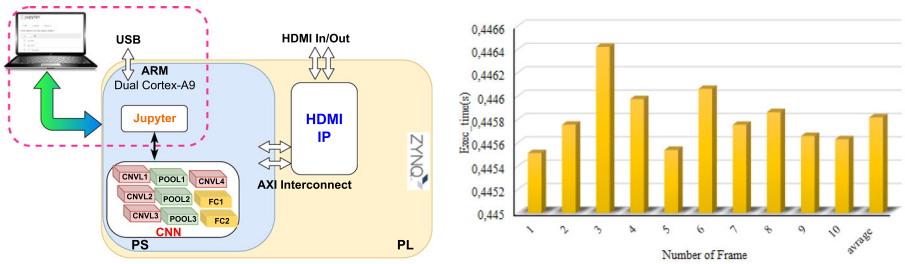


(a) Total Resources Vs. Reconfigurable Partition Resources



(b) IPs Hardware Cost Vs. Reconfigurable Partition Resources

Fig. 6 PYNQ-Z1 and reconfigurable partition resources



(a) ARM-based Software Implementation (b) Execution time

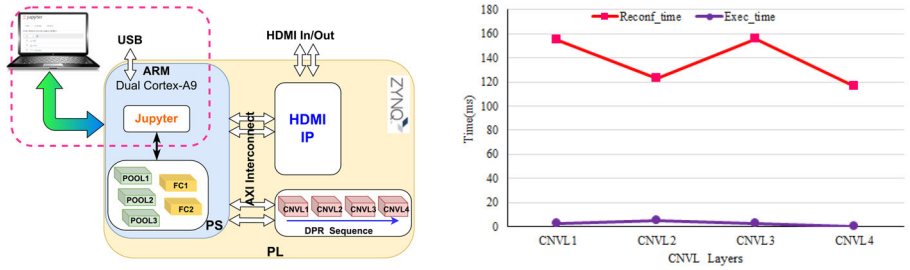
Fig. 7 Execution time of the ARM-based software implementation

software implementation is about 440 ms. This result is not sufficient and do not support real time-based deep CNN application. Based on this, we proposed to test the vision system by following another strategy.

This strategy, denoted by Fig. 8a, aims at implementing CNVL IP cores into the PL part using the DPR technique. while the pooling layer, fully connected layers, and activation functions are implemented the PS side. We evaluate the execution time for each CVNL IP in the PL side, while comparing this time to the reconfiguration time for each CNVL IP core, as presented by Fig. 8b. However, to process one image we should reconfigure 4 times the system using the DPR technique, beginning by the CNVL1 IP then the CNVL2 until reach the CNVL4 IP, without forgetting the PS processing between CNVL layers as well as FC, Pool, and activation functions. It is remarkable that in comparison to the run time of CNVL's HW implementations, the reconfiguration time overhead is shockingly too great. We note that the mean HW execution time of one CNVL IP core is over than 2.6 ms compared to over than 137 ms as a mean reconfiguration time. Therefore, the total execution and reconfiguration time of this strategy for all CNVL IP cores exceed 500 ms if we if we add up the mean reconfiguration and execution time of all IP cores. In addition, by evaluating the run and reconfiguration time ratio (mean ratio) represented by Fig. 8c, which is over about 5 ms for each CNVL IP cores, we note that this strategy cannot overcome the SW solution. On the other hand, Fig. 8d presents the total power consumption at each reconfiguration. The total power consumption is about 1.26 W when reconfigure the design by the CNVL1, when it is about 1.201 W when reconfigure the design by CNVL2. In addition, the total power continues the decrease when implement the CNVL3 which is around 1.18 W which is not the same when implement CNVL4, which is about 1.128 W. This is due to the difference in hardware cost occupied by each CNVL IP cores. Altogether, the CNVL IPs cores-based sequential reconfiguration would result in a high CNN inference execution time than the CNN regarded as a SW solution on the PYNQ-Z1 ARM dual core. In this context, we proposed another implementation strategy exploiting the DPR.

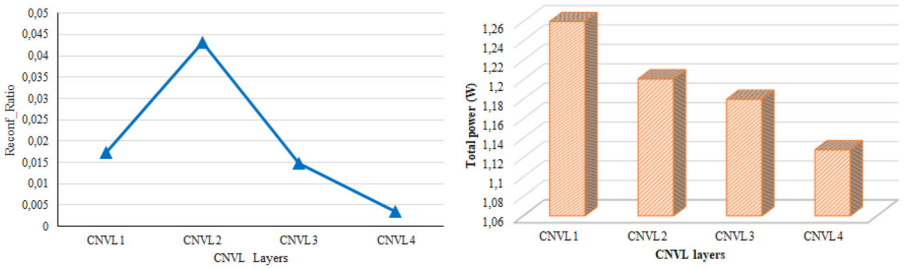
After the first strategy, that proposes to implement the CNN accelerator in the embedded PYNQ-Z1 ARM, the second strategy suggest to move the CNVL IPs cores on the CNN to the PL parts and exploits the DPR technique. Based on the result, we conclude that there is still room for improvement. At this stage, the third implementation strategy proposes to make choice on the CNVL IP's cores that will be moved to the PL based on their required time for the reconfiguration. Is it remarkable from previous results that the CNVL2 and CNVL4 present the minimum required time for the reconfiguration on the PL. In addition, CNVL1 and CNVL3 present the maximum required time for the reconfiguration. On the other hand, we can consider also execution time as choice factor to choose the CNVL IP's core that will be reconfigured on the PL part. Thus, will give the CNVL3 and CNVL4 IPs as stored the minimum execution time. But this solution will return to the first second strategy which is the sequential reconfiguration that led to increase the reconfiguration time. Therefore, the reconfiguration time is the best factor that can will be used in this implementation strategy with the aim is to accelerate the processing.

However, two sub-strategies, as denoted in Fig. 9, that provides the design configuration strategy and its results. Figure 9a provides the first configuration in this implementation strategy, which shows that the HDMI IP and the reconfigurable CNVL1 IP and CNVL3 are moved to the PL side, while FC and pool layers and activation functions are implemented in the PS side. The results of this implementation technique and test on ten images, as presented in Fig. 9b, prove that this method achieved the processing acceleration, which is about 0.29 s, compared to the previous software implementation (0.44 s). This acceleration can speed up the software run time by 1.5 times. On the other hand, the second design configuration that moves CNVL2 and CNVL4 IPs cores to the PL side and implements the other layers in the PS side, is represented by Fig. 9c. Therefore, the results denoted in Fig. 9d prove that the proposed implementation strategy is well-performed and the acceleration is reached by speeding up the software implementation by 2.3 times which make it the best implementation strategy using co-designing and DPR techniques.



(a) PYNQ-Z1 based HW/SW Implementation

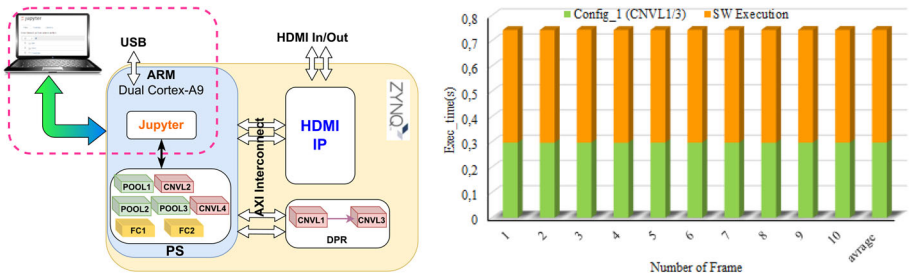
(b) HW Execution time Vs. Reconfiguration time



(c) The Execution and Reconfiguration time ratio

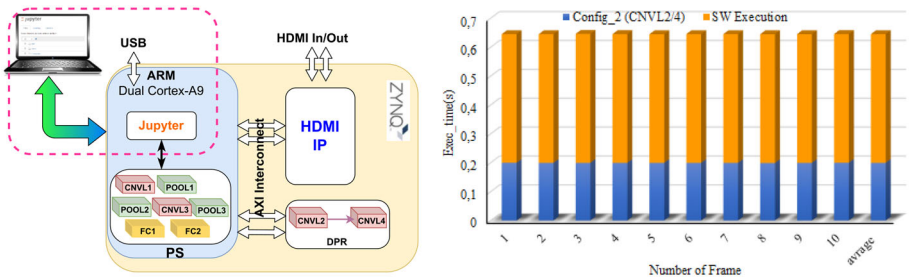
(d) The Total power consumption at each CNVL configuration

Fig. 8 Execution and reconfiguration time and the total power of the PYNQ-Z1 based HW/SW implementation



(a) Design-based CNVL1 and CNVL3 Configuration.

(b) HW execution time Vs. SW execution time.



(c) Design-based CNVL2 and CNVL4 Configuration

(d) HW execution time Vs. SW execution time.

Fig. 9 Execution time of the PYNQ-Z1-based HW/SW implementation of configuration 1 and 2

7.2 Comparative study

On the other hand and to more evaluate our proposed work to recent works, supplementary material of table 2 highlights the comparative results for our work, and the works cited in [11, 23, 26–28]. However, the work in [26] tries to find the best trade-off between high speed DNN and low power consumption. Hence, the authors designed several DNN topologies to ultimately select the best one. First, we note that we have not used the same FPGA family of Xilinx for different topics but based on deep CNN with different architectures (layers and parameters sizes). Our proposed work outperforms the work cited in [26], in terms of hardware cost and power consumption but it does not reach a

best execution time. This is refers to the fact that the limited PR resources that limit the pipelining optimization for our design. The work proposed in [27] proposes a uniform design to accelerate both 2D and 3D CNNs topologies. By comparing our results with the results provided by this work, we remark that our proposed work achieves a best performances in terms of hardware cost, power consumption, and almost the same execution time. Regarding the work proposed in [28], authors in their work aimed at addressing the utilization degradation issue that occurred in a general convolution engine. They implemented many network topologies on the platform ZYNQ xc7z035 FPGA, among these architecture we found the MobileNet implementation with a higher working frequency of about 200MHz than our work.

On the other side, our DPR-based CNN design outperformed this work by achieving the minimum consumption power and hardware cost. Finally, our work achieved best results in terms of hardware cost and power consumption to the other two works, cited in [11, 23], that attempt to propose a co-design based on accelerated CNN architectures for embedded vision applications. As a conclusion, our designs were able to achieve a reasonable balance between resource cost, power consumption, reconfiguration, and execution time, making it appropriate for applications on embedded devices with constrained resource budgets.

8 Conclusion

This paper proposes a DPR architecture for DCNN accelerator implemented on FPGA-SoC for face point detection. We proposed to accelerate each convolutional layer of DCNN for embedded facial point detection using Vivado HLS tools that is implemented on FPGA-SoC. We proposed a GPU-based software implementation for DCNN-based face point detection. Regarding to these results, the CNN F1 level is selected for the acceleration process. After that, a CNN-based acceleration and optimization method using Xilinx Vivado HLS tool is developed. Then, a DPR-based hybrid architecture is created to improve the proposed approach performances. Therefore, we proposed many scenario to evaluate our proposed work beginning by the FPGA-SoC-based ARM implementation to the reconfiguration by IP cores selection. Lastly, the obtained results showed that the proposed approach achieves higher performance in terms of reconfiguration time, execution time, hardware cost, and power consumption in comparison with other state-of-the-art implementations on FPGAs. As a future work, the quantization technique and the new Vitis tool of Xilinx will be used to exploit the new deep learning processor on the new embedded platforms.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11760-024-03177-2.

Acknowledgements The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through large group Research Project under Grant Number RGP.2/408/44.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Sun, Y., Wang, X., Tang, X.: Deep convolutional network cascade for facial point detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3476–3483 (2013)
2. Messaoud, S., Bradai, A., Bukhari, S.H.R., Quang, P.T.A., Ahmed, O.B., Atri, M.: A survey on machine learning in the internet of things: algorithms, strategies, and applications. Internet Things **12**, 100314 (2020)
3. Xu, X., Jiang, X., Ma, C., Du, P., Li, X., Lv, S., Yu, L., Ni, Q., Chen, Y., Su, J., et al.: A deep learning system to screen novel coronavirus disease 2019 pneumonia. Engineering **6**(10), 1122–1129 (2020)
4. Adel, E., Magdy, R., Mohamed, S., Mamdouh, M., El Mandouh, E., Mostafa, H.: Accelerating deep neural networks using fpga. In: 2018 30th International Conference on Microelectronics (ICM), pp. 176–179. IEEE (2018)
5. Bouaafia, S., Messaoud, S., Maraoui, A., Ammari, A.C., Khriji, L., Machhout, M.: Deep pre-trained models for computer vision applications: traffic sign recognition. In: 2021 18th International Multi-Conference on Systems, Signals & Devices (SSD), pp. 23–28. IEEE (2021)
6. Yao, Y., Duan, Q., Zhang, Z., Gao, J., Wang, J., Yang, M., Tao, X., Lai, J.: A fpga-based hardware accelerator for multiple convolutional neural networks. In: 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp. 1–3. IEEE (2018)
7. Lahmyed, R., Ansari, M.E., Kerkaou, Z.: Automatic road sign detection and recognition based on neural network. Soft Comput. **26**, 1743–1764 (2022)
8. Guo, J., Yin, S., Ouyang, P., Liu, L., Wei, S.: Bit-width based resource partitioning for cnn acceleration on fpga. In: 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 31–31. IEEE (2017)
9. Messaoud, S., Bouaafia, S., Maraoui, A., Khriji, L., Ammari, A.C., Machhout, M., et al.: Virtual healthcare center for covid-19 patient detection based on artificial intelligence approaches. Can. J. Infect. Dis. Med. Microbiol. 2022 (2022)
10. Wu, D., Zhang, Y., Jia, X., Tian, L., Li, T., Sui, L., Xie, D., Shan, Y.: A high-performance cnn processor based on fpga for mobilenets. In: 2019 29th International Conference on Field Programmable Logic and Applications (FPL), pp. 136–143. IEEE (2019)

11. Bouaafia, S., Khemiri, R., Messaoud, S., Sayadi, F.E.: Deep cnn codesign for hevc cu partition prediction on fpga-soc. Neural Process. Lett. **54**(4), 3283–3301 (2022)
12. Chen, Y., Xie, Y., Song, L., Chen, F., Tang, T.: A survey of accelerator architectures for deep neural networks. Engineering **6**(3), 264–274 (2020)
13. Guo, K., Zeng, S., Yu, J., Wang, Y., Yang, H.: A survey of fpga-based neural network accelerator. arXiv:1712.08934 (2017)
14. Maraoui, A., Messaoud, S., Bouaafia, S., Ammari, A.C., Khriji, L., Machhout, M.: Pynq fpga hardware implementation of lenet-5-based traffic sign recognition application. In: 2021 18th International Multi-Conference on Systems, Signals & Devices (SSD), pp. 1004–1009. IEEE (2021)
15. Ahmad, A., Al Busaidi, S.S., Al Maashri, A., Awadalla, M., Hussain, S.: Fpgas-chronological developments and challenge. Int. J. Electr. Eng. Technol. **12**(11), 60–72 (2021)
16. Fukushima, Y., Iizuka, K., Amano, H.: Parallel implementation of cnn on multi-fpga cluster. IEICE Trans. Inf. Syst. **106**(7), 1198–1208 (2023)
17. Shehzad, F., Rashid, M., Sinky, M.H., Alotaibi, S.S., Zia, M.Y.I.: A scalable system-on-chip acceleration for deep neural networks. IEEE Access **9**, 95412–95426 (2021)
18. Heidaryan, M., et al.: Fpga implementation of two multilayer perceptron neural network in cascade for efficient real time hand gestures tracking. Microprocess. Microsyst. **100**, 104849 (2023)
19. Xiao, Q., Liang, Y.: Fune: an fpga tuning framework for cnn acceleration. IEEE Design Test **37**(1), 46–55 (2019)
20. Kästner, F., Janßen, B., Kautz, F., Hübner, M., Corradi, G.: Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 154–161. IEEE (2018)
21. Brownlee, J.: How to use learning curves to diagnose machine learning model performance, 2019 (2021)
22. Huang, L., Li, D.-L., Wang, K.-P., Gao, T., Tavares, A.: A survey on performance optimization of high-level synthesis tools. J. Comput. Sci. Technol. **35**, 697–720 (2020)
23. Messaoud, S., Bouaafia, S., Maraoui, A., Ammari, A.C., Khriji, L., Machhout, M.: Deep convolutional neural networks-based hardware–software on-chip system for computer vision application. Comput. Electr. Eng. **98**, 107671 (2022)
24. Vallina, F.M.: Implementing memory structures for video processing in the vivado hls tool. XAPP793 (v1.0), vol. 20 (2012)
25. Hassan, A., Ahmed, R., Mostafa, H., Fahmy, H.A., Hussien, A.: Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on fpga. In: 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), pp. 183–186. IEEE (2015)
26. Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., Li, J.: An fpga-based cnn accelerator integrating depthwise separable convolution. Electronics **8**(3), 281 (2019)
27. Liu, Z., Chow, P., Xu, J., Jiang, J., Dou, Y., Zhou, J.: A uniform architecture design for accelerating 2d and 3d cnns on fpgas. Electronics **8**(1), 65 (2019)
28. Zhang, N., Wei, X., Chen, H., Liu, W.: Fpga implementation for cnn-based optical remote sensing object detection. Electronics **10**(3), 282 (2021)