



HAL
open science

On building a CNN-based multi-view smart camera for real-time object detection

Jonathan Bonnard, K. Abdelouahab, Maxime Pelcat, F. Berry

► To cite this version:

Jonathan Bonnard, K. Abdelouahab, Maxime Pelcat, F. Berry. On building a CNN-based multi-view smart camera for real-time object detection. *Microprocessors and Microsystems: Embedded Hardware Design*, 2020, 77, pp.103177. 10.1016/j.micpro.2020.103177 . hal-02931775

HAL Id: hal-02931775

<https://hal.science/hal-02931775v1>

Submitted on 10 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On Building a CNN-based Multi-view Smart Camera for Real-time Object Detection

Jonathan Bonnard¹, Kamel Abdelouahab², Maxime Pelcat^{1,3}, François Berry¹

Abstract

Multi-view image sensing is currently gaining momentum, fostered by new applications such as autonomous vehicles and self-propelled robots. In this paper, we prototype and evaluate a multi-view smart vision system for object recognition. The system exploits an optimized Multi-View Convolutional Neural Network (MVCNN) in which the processing is distributed among several sensors (heads) and a camera body.

The main challenge for designing such a system comes from the computationally expensive workload of real-time MVCNNs which is difficult to support with embedded processing and high frame rates. This paper focuses on the decisions to be taken for distributing an MVCNN on the camera heads, each camera head embedding a Field-Programmable Gate Array (FPGA) for processing images on the stream. In particular, we show that the first layer of the AlexNet network can be processed at the nearest of the sensors, by performing a Direct Hardware Mapping (DHM) using a dataflow model of computation.

The feature maps produced by the first layers are merged and processed by a camera central processing node that executes the remaining layers. The proposed system exploits state-of-the-art deep learning optimization methods, such as parameter removing and data quantization. We demonstrate that accuracy drops caused by these optimizations can be compensated by the multi-view nature of the captured information.

Experimental results conducted with the AlexNet CNN show that the proposed partitioning and resulting optimizations can fit the first layer of the multi-view network in low-end FPGAs. Among all the tested configurations, we propose 2 setups with an equivalent accuracy compared to the original network on the ModelNet40 dataset. The first one is composed of 4 cameras based on a Cyclone III E120 FPGA to embed the least expensive version in terms of logic resources while the second version requires 2 cameras based on a Cyclone 10 GX220 FPGA. This distributed computing with workload reduction is demonstrated to be a practical solution when building a real-time multi-view smart

^{*}Sources files available online: <https://github.com/DreamIP/mvcnn>

¹UCA - Institut Pascal, UMR CNRS 6602, Clermont-Ferrand, France

²Sma-RTy SAS - Clermont-Ferrand, France

³Univ Rennes, INSA Rennes, IETR, UMR CNRS 6164, Rennes France

camera processing several frames per second.

Keywords: CNN, FPGA, MVCNN, Smart Camera, Real-Time

1. Introduction

Multi-view vision systems can address several challenges in computer vision such as 3D reconstruction, depth map sensing and object classification [1, 2, 3]. Significant progress have been made in the domain of object recognition by using deep learning algorithms, and more specifically, Convolutional Neural Networks (CNNs) that constitute the state of the art algorithms since Krizhevsky et al. [4] achieved a top-5 test error below 20% in the 2012 ImageNet competition. Several major contributions to deep learning have introduced significant changes with respect to the first AlexNet model, proposing deeper networks [5, 6, 7] to improve recognition rate and architectures that are more suited for the computational power of embedded systems [8, 9]. In addition, Multi-View Convolutional Neural Networks (MVCNNs) have recently emerged, motivated by the fact that additional features extracted from multiple point of views of an object (e.g. depth maps or 3D points coordinates) provide more representative data on a scene than a single view flat image. In a multi-view context, previous publications demonstrate that state of the art CNNs become more accurate in terms of object classification when exposed to multiple views [10, 11, 12, 13, 14, 15].

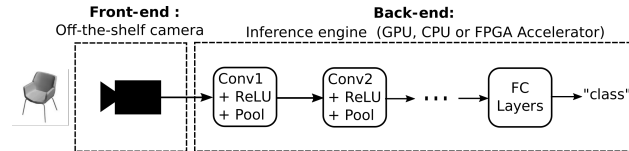
In both single-view and multi-view CNNs, training and inference tasks are typically performed by Graphics Processing Units (GPUs) which are faster by several orders of magnitude when compared to conventional CPUs for the same level of programming complexity (eg. using the CUDA or OpenCL programming languages). In the mean time, Field-Programmable Gate Arrays (FPGAs) are garnering increasing attention for accelerating machine learning algorithms. Indeed, the massively parallel structure of FPGAs, their low power consumption and their low and controllable processing latency, are making them top candidates for CNN inference, even in a context of cloud computing [16]. However, the main challenge of FPGA-based CNN inference comes from the millions of weights that need to be stored in on-chip memory (e.g. 6.8M weights for the GoogLeNet CNN). To tackle with this challenge, researchers have introduced different approximate computing methods including the quantization of convolutional weights [17], the reduction of operand bit-width down to 2 bits[18], the introduction of accelerators based on SIMD architectures [19], some hardware/software hybrid solutions built from OpenCL [20, 21, 22], or full hardware solutions using high level synthesis [23]. These studies have shown that FPGAs are competitive platforms for CNN inference and can be optimized to simplify their implementations with reduced hardware resource utilization. Still, the Direct Hardware Mapping (DHM) of a CNN to an FPGA [24], i.e. the one-to-one translation of neurons to hardware without time multiplexing, of an entire state-of-the-art CNN onto an FPGA, remains impossible even on the largest

FPGA devices, although it is the most performing use of the FPGA structure, optimizing data locality to the limit. As a consequence, when targeting DHM, methods are needed to split processing between multiple systems and fit the sub-networks into several sub-systems [25]. On the particular problem of physical object recognition from multiple camera-captured views, this method can be particularly appropriate, as the processing of different sources of data can be delegated to different front-end sub-networks, accelerated close to their respective sensors.

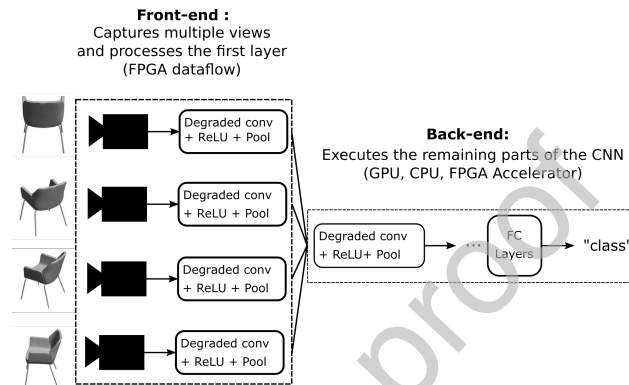
In this context, the present study aims at applying multi-view deep learning, as introduced in [10], within an environment composed of a set of synchronized smart cameras based on Field-Programmable Gate Array (FPGA) and a *back-end sub-system which can be either an FPGA, a CPU or a GPU, as depicted in figure 1*. The main contribution of the paper is a design method for multi-view vision systems that jointly optimizes the multi-system porting of the computational workload, and the accuracy of the implemented vision task. The resulting multi-view camera combines front-end near-sensor processing of the first CNN layer(s) and a back-end processing of the remaining CNN layers. A real prototype of the camera smart heads is built and assessed. Using the advanced diversity offered by multiple views, we demonstrate that the built CNN, in spite of a front-end/back-end decomposition and network optimizations and degradation, *is equivalent to a monocular CNN*, but is also able to deliver real-time performance for our camera setup⁴ thanks to the DHM of convolutions and to workload reduction.

The paper is organized as follows :

- Section 2 gives an overview of previous studies in multi-view object recognition with CNNs. *In a second part, the two main FPGA-oriented computation paradigms of CNNs are introduced, which both are exploited for building the multi-view camera system*
- Section 3 details the proposed multi-view vision system design method,
- Section 4 describes the conducted optimizations of AlexNet architecture in this multi-view context,
- Section 5 reports the performance of the proposed method on the built prototype system.



(a) Single view camera setup with real-time inference



(b) Proposition : Multi-view camera setup with equivalent real-time inference

Figure 1: Figure (a) depicts a single-view camera capturing a scene and a back-end performing the full computation of a mono-view CNN for object recognition. In Figure (b), the proposed evolution introduces 4 synchronized smart camera heads as part of a multi-view smart camera, providing different points of view while pre-processing CNN data in a distributed way. A front-end network is severed from the main CNN and sliced between the camera heads. Using standard labelling, Conv designates convolutional layers, ReLU designates non-linear functions, Pool designates data dimensional reduction and FC refers to the fully connected, non-convolutional layers used for classification.

2. Related works

2.1. Multi-View Object Recognition with CNN

While classical inference CNNs work on a single view (figure 2), methods have recently been proposed to process 3D data representing objects with synthetic 3D meshes [13], depth maps [26], point clouds [27] and multiple views. In particular, multi-view networks are gaining an increasing research interest, particularly for classifying and retrieving 3D shapes⁵.

The first study related to multi-view CNN shape recognition was conducted by *Su et al.* [10]. It relies on a graph depicted in figure 3. In this scheme, several virtual cameras surround an object to get different view points and each

⁴In this work, we are aiming at a real-time performance of 30 Frames Per Second (FPS) with up to 4 views.

⁵In this paper, the "multi-view" term refers to views from different points of view, not from multiple scales

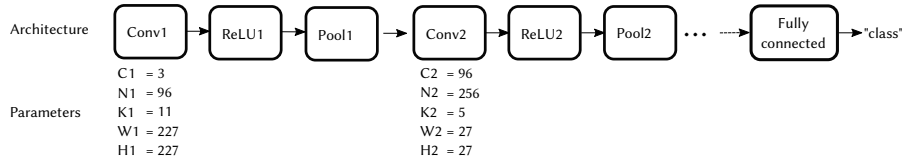


Figure 2: Conventional architecture of the AlexNet CNN. The first layer is composed of N_1 convolutions acting on the 3 color planes (C_1) of the input frame with a kernel size of K_1 to extract different features. ReLU decides whether the input of a neuron (feature maps) is relevant or not for the current detection. Then, the pooling layer downsizes the feature map to let the next layers detect larger features in the original image.

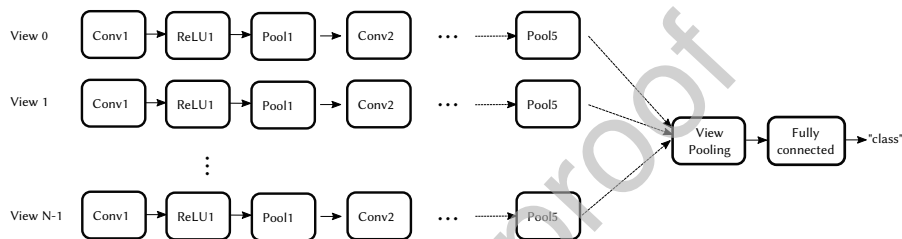


Figure 3: The multi-view setup introduced by *Su et al.*

view is coupled to the convolutional part of a CNN where deep feature maps are regrouped into a single one by a *view pooling* layer. View pooling operates similarly to a usual max pooling, keeping the maximum value out of a set of inputs, with the difference that the maximum value is no longer that of the direct local neighborhood of a $N \times N$ window of activations, but the maximum pixel to pixel value of each view. The following (fully connected) layers perform a classification equivalently to a single view CNN and estimate the class of the object. The gain in classification rate reaches 4.8% with 12 cameras compared to a VGG-M CNN using a single view.

Feng *et al.* [11] have demonstrated that each view perspective in a multi-view CNN contributes differently to the shape descriptor. As a result, authors advocate to classify perspectives according to their degree of importance in the description of the object to be classified. While the authors' proposed network adds some computational overhead due to the grouping module, this architecture is more accurate by 3.2% compared to the multi-view CNN mentioned above. Kanezaki *et al.* [15] propose a solution for estimating the pose of an object based on a multi-view framework similar to the previous studies. Pose estimation is performed in an unsupervised manner and classification is performed at the same time. This training method considers the view points as variables optimized during the training sequence. This work reaches the same performance level as a 80 view MVCNN while requiring only a few different point of views of an object.

These studies have demonstrated that multi-view architectures can bring performance improvement over traditional networks for recognizing objects. This

110 performance is however obtained at the cost of more computational resources,
 as the convolution section of a network is duplicated as many times as there are
 views in the system or views are batched into the memory of a single inference
 system (GPU, CPU or FPGA accelerator) with the consequence of increasing
 115 inference time. The camera setup we propose in the current study follows a
 different multi-view inference strategy. The design of embedded smart cameras
 very quickly shows the limits of current computational technologies in terms of
 convolutional network computation capabilities, especially when this inference
 is performed on low-end FPGAs. These limits force the designer to optimize the
 CNN architecture, for instance by decreasing the number of features, but these
 120 optimizations impact classification quality. Building on the work mentioned
 above, i.e. on capturing an object from several viewpoints in order to increase
 the rate of classification, we propose a method to jointly optimize the number
 of features, convolution kernels and the quantification of data, and to map one
 or more front-end layers on a CNN into FPGAs using a dataflow representation
 125 of the network.

In order to fit object classification into the built multi-view smart camera,
 the study has consisted in separating a front-end and a back-end network and
 optimizing the resources needed by front-end networks implemented near their
 130 respective sensors. The CNN features that have been deeply modified are:

- the number of feature maps within all layers that has been drastically reduced,
- the size of the kernels in the front-end convolutional layer that has been strongly reduced.

135 These two operations have a negative impact on classifier performance.
 Nonetheless, this study demonstrates that the multiple views acquired by our
 camera system compensate most of the detection degradation due to network
 optimizations. The novelty of this work comes from the Multi-view camera op-
 timization that manages, by combining state-of-the-art methods, to fit real-time
 140 deep learning in an FPGA-based multi-view camera. As a result, workload can
 be balanced between the front-end and the back-end in order to accelerate the
 computation.

2.2. CNN inference on FPGAs

As highlighted in [28], convolution layers are the most computationally in-
 145 tensive parts of a CNN. Indeed, the execution of these layers requires many
 Multiply Accumulate (MAC) operations. The number of MACs occurring on a
 convolution layer is expressed in eq.1,

$$\mathcal{W} = W \times H \times N \times C \times K \times K \quad (\text{MAC}) \quad (1)$$

where N is the number of output feature maps, (W, H) the width and height of
 output feature maps, C the number of input feature maps and $K \times K$ the size
 150 of the convolution kernels.

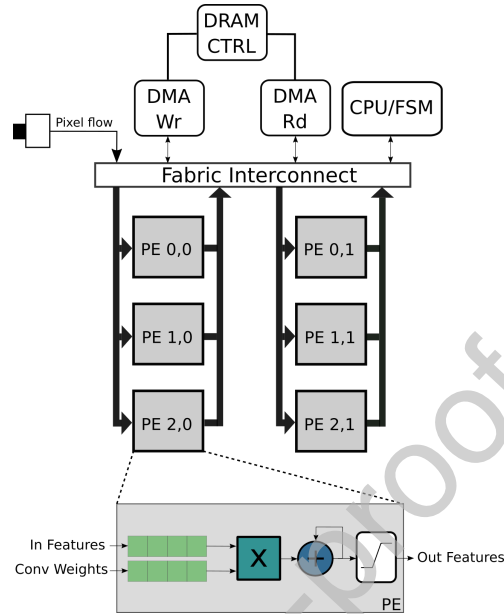


Figure 4: A simplified view of a CNN accelerator architecture on FPGA. A CPU or a specialized finite state machine schedules the overall layer computations of a CNN with a set of specialized Processing Elements (PEs) performing layers' computation. A direct memory access (DMA) engine fetches input feature maps and convolution weights into PEs while another DMA stores output features into an external memory bank.

Several studies leverage on the FPGA computational power to accelerate the execution of convolution layers [29]. In particular, two computational paradigms exist [30]:

- On one side of the spectrum, *computation engines* (figure 4) are widely present in the literature [31, 32, 16], and support a large variety of CNN workloads. Computation engines are based on a fixed architecture that typically takes the form of a systolic array of Processing Elements (PEs). In the case of convolution layers, each operation corresponds to a sequence of micro-instructions executed by the hardware accelerator. The limitations of such architectures come from 2 factors: the number of available PEs that execute convolutions, and the external memory latency to fetch/store pixels, convolutional weights, and processed data as pointed out in [33]. However, these processing engines have a good flexibility and support a large variety of CNN models.
- On the other side of the spectrum, *dataflow* architectures with Direct Hardware Mapping (DHM) [34, 35, 24] map one distinct hardware block for each layer of the target CNN. The blocks are then pipelined for them to process data in a *streaming* fashion. As a consequence, dataflow CNN accelerators deliver an overall better performance at the price of a lower

170 flexibility. As will be detailed in the next section, the multi-view system of the current study leverages a dataflow architecture, and aims to rationalize use of the resources available in a pool of smart camera heads based on low-end FPGAs, so as to achieve real-time performance (30 FPS) for the object recognition task.

175 3. Technical Choices when Building the Camera

As shown in the figure 5, the inference model of a single view CNN (2) is described as the composition of two functions H and FC . H represents the entire convolutional section of the CNN extracting the features from the image of a camera. FC is the fully connected layer performing classification.

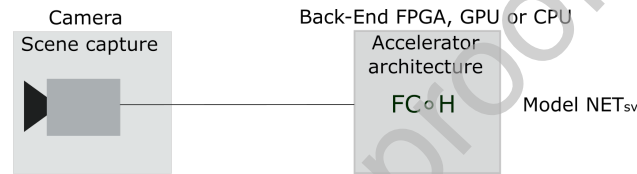


Figure 5: Generic CNN model of inference with accelerator with a single camera

$$NET_{sv} = FC \circ H(input(camera)) \quad (2)$$

180 The multi-view architecture model of *Su et al* [36] assumes that several V views are concurrently processed by the convolution and pooling parts of the MVCNN (figure 3). Next, a view-pooling layer gathers data from all views and forwards them to the rest of the CNN for classification. This model can be written as :

$$NET_{MVCNN} = FC \circ G \circ H_v(input(camera_v)), \quad 1 < v < V \quad (3)$$

185 where H_v represents the entire convolution section of a CNN which is duplicated in respect to the number of views v and G refers to the view-pooling layer introduced in the previous section.

This model provides better classification accuracy when compared to a single-view setup as the number of available view points increases. However, MVCNNs 190 are prohibitive and inefficient for embedded systems since the whole convolutional part of the network has to be processed independently for each view before the classification stage.

The method presented in this paper is a modified version of this model. First, it introduces a splitting parameter m that divides a network into two sub-networks: A front-end (F^m) that implements the first CNN layers on the camera head and a back-end (H^{M-m+1}) that executes the remaining layers. 195 This can be formalized as:

$$NET_{sv} = FC \circ H^{M-m+1} \circ F^m \quad (4)$$

200 In a context of multi-view system, the previous model can be extended by duplicating the front-end sub-network across V smart-camera heads. In this case, the view-pooling function (G) merges the features from the front-end and feeds the back-end, as written in equation 5:

$$NET_{MVCNN} = FC \circ H^{M-m+1} \circ G \circ F_v^m \quad \text{with } 1 < v < V \quad (5)$$

205 With the formalism introduced in Eq.5, the front-end layers are added without any additional latency cost, as they are processed in parallel with the previously evoked DHM strategy. Conversely, back-end layers are shared by all views and do not cost more than the respective layers in a single-view CNN. Note 210 that the value of the parameter m greatly impacts the computational workload of a given multi-view CNN. Indeed, the smaller this parameter is, the less layers will be replicated and, consequently, the lower the computational workload will be. However, low-end FPGA devices such as the ones employed in the camera heads still can not map the front-end network, sometimes not even the first 215 layer when $m = 1$ [37]. In this case, the cost of layer F^1 in terms of multipliers is given by Equation(6).

$$\text{Cost}_{hw}(F^1) = N_1 \times C_1 \times K_1^2 \quad (\text{Multipliers}) \quad (6)$$

220 To tackle this problem, a possible solution is to reduce the number of parameters of the front-end and back-end. The derived degraded sub-networks, respectively \hat{F}^m and \hat{H}^{M-m+1} , result in an accuracy drop that aims to be compensated with multi-view nature of the input data. The degraded model can be written as:

$$NET_{MVCNN} = FC \circ \hat{H}^{M-m+1} \circ G \circ \hat{F}_v^m \quad \text{with } 1 < v < V \quad (7)$$

225 As a proof of concept, a prototype of multi-view camera is built based on the same architecture as a state of the art AlexNet CNN as depicted in figure 6. Then, the number of the CNN parameters is gradually decreased in order to fit front end networks into near-sensor FPGAs. This work only considers the 230 first layer of this CNN as a front-end ($m = 1$). By decreasing the number of operations by a factor d in the front-end part of the camera, it becomes possible to embed all its operations following a dataflow DHM partitioning method, as shown in 8.

$$\text{Cost}_{hw}(\hat{F}^1) = d \times N_1 \times C_1 \times K_1^2 \ll \text{Cost}_{hw}(F^1), \forall d \in]0; 1[\quad (8)$$

235 In the next sub-sections, we detail the introduced optimizations and the corresponding hardware architecture.

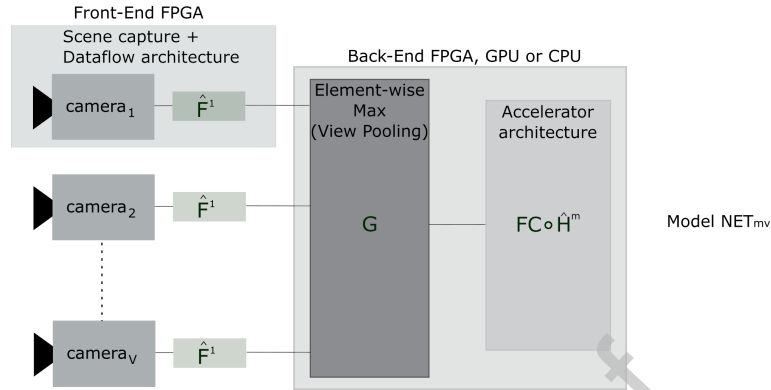


Figure 6: CNN model distributed between multiple smart cameras with dataflow model of computation and an accelerator back-end

3.1. Near-sensor Hardware Acceleration through Direct Hardware Mapping

DHM takes advantage of the data-path nature of CNNs feed forward propagation and leverages a representation of the CNN with a dataflow model of computation. In this paradigm, the CNN is modelled as a dataflow graph wherein the nodes correspond to processing actors and the edges correspond to communication channels. Each actor follows a purely data-driven execution model where execution is triggered only by the availability of input operands [38]. The resulting dataflow accelerators infer one distinct hardware block for each actor of the CNN as illustrated in Figure 7. By taking this paradigm to the extreme, the Direct Hardware Mapping (DHM) method consists of *physically* mapping the whole graph of actors onto the target device [24] and delivers new outputs on each clock cycle. As a result, each of the operations involved in a CNN layer becomes a computing unit with its specific hardware instance on the FPGA. In the case of convolution layers, DHM fully *unrolls* the processing of MAC operations producing N results per clock cycle⁶. To deliver this throughput, a dataflow accelerator physically maps the amount of multipliers described in equation 6.

As pointed-out in [24], dataflow architectures with direct hardware mapping catch all the parallelism patterns exhibited by CNNs, resulting in a high computational throughput. Moreover, DHM exempts from the main constraints limiting the performance of embedded CNN accelerators as weights and intermediate data do not require to be fetched and stored in external memory. However, the major downside of DHM is related to the scalability of the accelerator. Indeed, a CNN mapping can rapidly be bounded by the available resources present in a given FPGA. In particular, DHM can not implement layers or networks that involve more multiplications than the number of multi-

⁶Thus, the dataflow accelerator requires $U \times W$ clock cycles to process the full layer

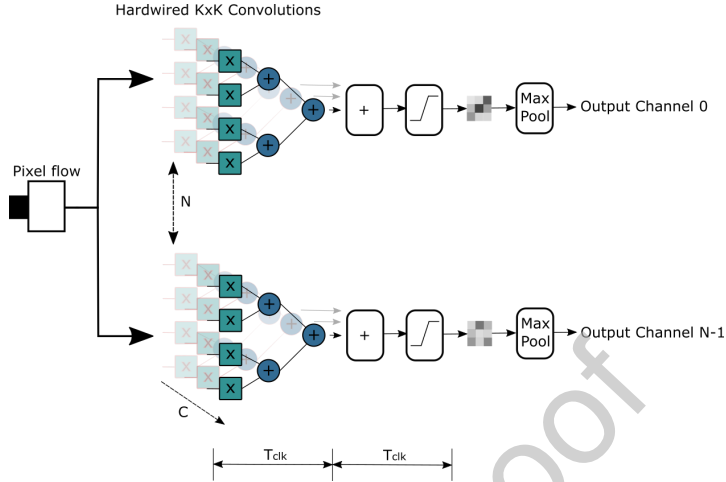


Figure 7: An example of the first layer directly mapped into $N \times C \times K \times K$ multipliers. The pipeline delivers data after $2 \times T_{clk}$

265 pliers available on a given FPGA. This is in contrast with the time-multiplexed computation engines, which favour scalability and flexibility over customization and computational throughput.

270 For the reasons evoked above, this work considers the two computation paradigms represented by the accelerated part \hat{H} and the dataflow part \hat{F} in the model (7). The first CNN layer \hat{F}^1 , which mapping tends to have the lowest resource requirements (Table 1), can be implemented in a DHM fashion using dataflow and reduced data representations⁷. By contrast, the deep layers \hat{H} , which tend to be more computationally intensive, can be implemented either on an FPGA device or an embedded GPU by means of a computation engine, as discussed in the previous section.

Alexnet (convolutional Layers)	L1	L2	L3	L4	L5
Multipliers (8bits)	35K	614K	885K	1327K	885K
Logic Elements (LEs)	800K	N.C	N.C	N.C	N.C

Table 1: Amount of multipliers and logic elements required for each layer of AlexNet CNN in a dataflow model of computation using 8-bit inputs, weights and output data.

275 3.2. Data quantization

The other limiting factor of CNN inference on FPGA comes from the data quantization used in state of the art networks. Considering the large number of

⁷Only the first layer could be synthesized with the Intel Quartus software. The other layers synthesis requested more than the 160GB of RAM resources available on our server

operations to perform and the amount of data to fetch and store into memory on each layer of a deep neural network, previous studies led to optimizations on data and weight quantization. This feature has been widely discussed in the literature, proposing to downsize the simple precision floating point (FP32) data representation to half precision floating point (FP16) [22], 8 bits floating points [16] and even 8 bits integer [39, 40] representations with low impact on classification accuracy, denoting a strong resistance to quantization noise of CNN.

This aspect is important on FPGA devices given the limited number of floating point Digital Signal Processing (DSP) blocks. Conversely, the number of hardwired 8-bit multipliers is significantly higher (each FP32 DSP block can implement up to three concurrent fixed point multiplications) and can be expanded further using the logic fabric of a FPGAs, especially through the direct hardware mapping strategy evoked above. In order to fit the first layer \hat{F}^1 of our CNN into the camera heads, this work exploits an 8-bit quantization of the convolution weights and output data. In addition to processing gains, such parameter size reduction impacts the computational energy consumption [41, 42] as the amount of data to transfer from a layer to its neighbour (or to external memory) is reduced.

3.3. Camera heads and their FPGA

Our multi-view system prototype (Fig. 8) is divided into two parts with a front-end system composed of multiple smart cameras heads and a back-end system as depicted in the figure 1b of section 2. The back-end is used as the central system, sending commands to the camera heads through a data-link, in our case a gigabit Ethernet communication layer. The back-end sends simultaneous trigger commands to the camera heads, each embedding an Intel Cyclone III FPGA. Each head processing system translates the command (setting image size, frame rate, etc.) to its attached global shutter CMOS sensor. Next, the FPGA converts the raw data to an RGB image which is down-scaled to match with the input of the CNN.

Each of the smart cameras heads executes the front-end layer \hat{F}^1 of the CNN with a dataflow DHM strategy. This means that the first layer is processed at the image sensor clock rate, delivering feature data at the same rate. The feature maps resulting from head embedded processing are streamed through raw Ethernet packets to the back-end system in charge of computing the remaining layers of the CNN. Next sections detail the method employed to deploy the CNN on the multi-view smart camera, as well as experimental results.

4. Proposed method to optimize CNN inference in the camera

In this section, the challenges encountered when building a multi-view smart camera with dataflow DHM are explained, as well as a proposed method to

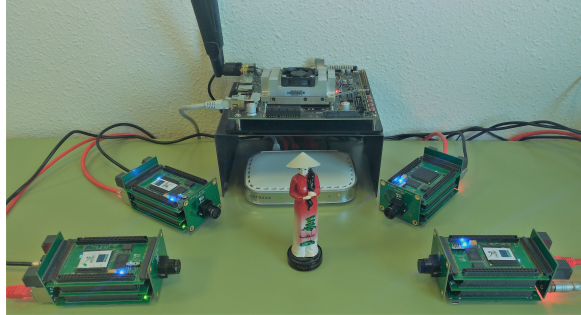


Figure 8: A setup example. The front-end camera heads (Cyclone III) and a test back-end connected through an Ethernet switch

320 tackle them. The proposed method iteratively reduces the number of channels and parameters. Indeed, network compression is widely advocated in the literature to speed-up the inference stage [43, 44]. As slimming networks affects classification performance, this study investigates the trade-off between network degradation and the multiplication of camera heads to maintain the original accuracy.

4.1. Dataset and training methodology

The training step of a specific CNN architecture for 3D shape retrieval or multi-view recognition requires a special database of CAD objects or a set of 2D images with attached intrinsic and extrinsic parameters of the camera capturing the object. For our study, we leverage the ModelNet40 dataset [26] that gathers 12k shapes from 40 different classes of objects in a mesh format. This format provides a set of points describing the volume of each object in the 3D space. The surface of each object can be rendered using either voxels, a Phong shading[45] or a ray tracing[46] algorithm to smooth the object shape and to give a more realistic visual perception. In this work, we used a balanced version of this dataset where each category has an equal number of object and views. The dataset comprises 3200 objects for training and 800 for validation. 12 virtual cameras are equally distributed around the Z-axis (one every 30 degrees) of the object with a pitch of 30 degrees (Figure 9). All of the CNN architecture variations presented in this paper are trained from scratch for 60 epochs using the Pytorch framework [47]. The initial learning rate for the Adam solver is set to 10^{-4} and the weight decay regularization parameter is set to 10^{-4} . For each 30 epochs, the learning rate is divided by 2.

4.2. Direct network compression

345 As highlighted in eq. 6, reducing the computational complexity of a convolutional layer involves modifying the on of two variables N , K or both, which respectively represent the number of generated features and the size of convolution kernels. As a first step, we reduced the number of features in the first

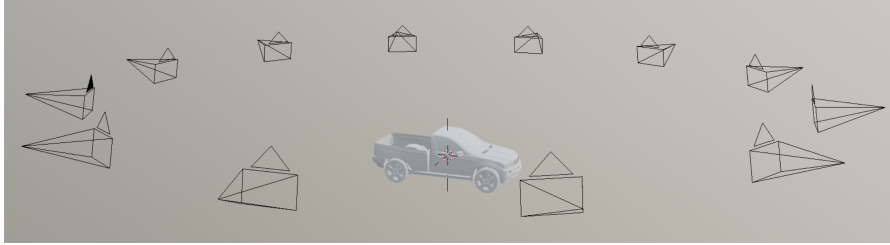


Figure 9: A 12-camera setup to render object files of the dataset

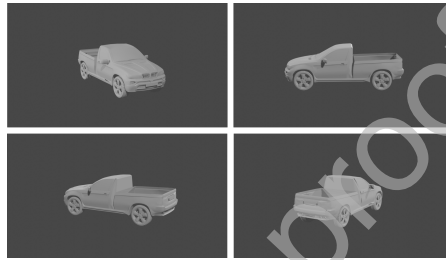


Figure 10: 2D views from the simulation of 4 cameras

layer of the CNN by lowering the parameter N_1 of the front-end⁸ F^1 by a factor d ($d \in [\frac{3}{4}; \frac{2}{3}; \frac{1}{2}]$). In parallel, the back-end of the network is set in two modes. The first one, namely H has the same number of parameters as the original AlexNet architecture with $(N_2, N_3, N_4, N_5) = (256, 384, 384, 256)$ convolution kernels. The second, \hat{H}^M has half the parameters with respectively (128, 192, 192, 128) output feature maps. In this second network architecture, we make the assumption that the combination of multiple viewpoints of the object will have a compensating effect on classification accuracy. More precisely, each feature extracts more informative features from the occluded parts of an object, so that the number of parameters N_i ($i \in [2; 3; 4; 5]$) can be reduced compared to the single view CNN.

For each new architecture, the network is trained on a subset of the ModelNet40 dataset, then the classification results on the validation subset are reported. We also report the accuracy of the trained networks on the ImageNet dataset for the least (\hat{H}^M) and the most (H^M) computationally expensive back-end architectures. This intends to outline the accuracy gap between architectures on different datasets. Figure 11(a) shows the network response to parameter reduction while maintaining a constant kernel size ($K_1 = 11$). The configurations for the first layer of the network are given on the x-axis according

⁸From now, F^1 represents the first layer of the original CNN and \hat{F}^1 refers to the first layer of the modified CNN.

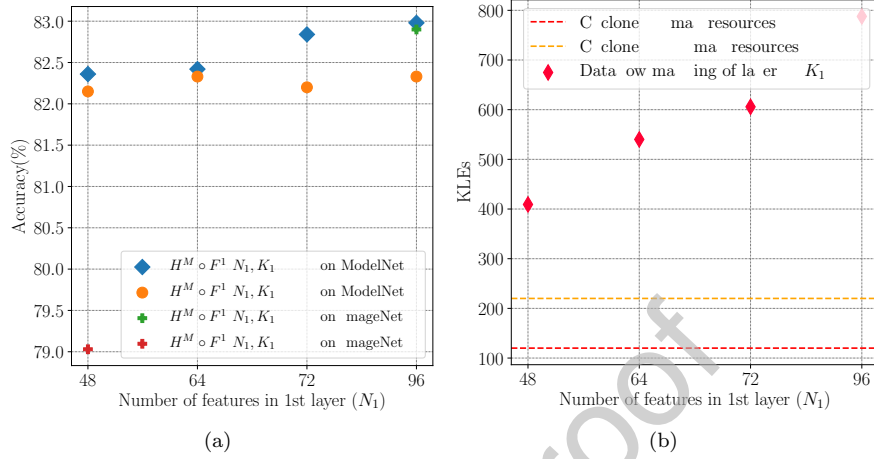


Figure 11: (a) The classification accuracy of 2 back-end networks (layers 2 to 5) and 4 different front-end setups (N_1). (b) The FPGA synthesis resources of the 4 front-end setups with a dataflow implementation.

to the combination of \hat{H}^M (half back-end) and H^M (full back-end). As shown in this figure, the maximum difference of performance between both architecture reaches 1.85% in terms of classification accuracy on the ModelNet40 dataset and 3.9% on ImageNet. The classification accuracy gap for the lightened architecture can be explained by the fact that a classifier requires much more parameters to solve ImageNet (1000 classes) than ModelNet40 (40 classes). As a conclusion, we decide to stop compressing the number of features of the network front-end \hat{F}^1 to $N_1 = 48$ and the back-end to \hat{H} to $(N_2, N_3, N_4, N_5) = (128, 192, 192, 128)$.

In terms of resource utilization, figure 11(b) reports the number of logic resources mapped by the synthesizer for the front-end \hat{F}^1 with a variable number of feature maps (N_1) and constant kernel size ($K_1 = 11$)⁹. As expected, the amount of resources grows linearly with the number of features, resulting in 400K logic elements mapped for the lightest architecture ($\hat{F}^1(N_1 = 48, K_1 = 11)$) and 800K for ($F^1(N_1 = 96, K_1 = 11)$). This resource utilization clearly exceeds the amount of logic elements made available on the targeted FPGAs (reported in dashed lines in the previous figure), which demonstrates that this layer should be further optimized by tuning the front-end convolution kernel size in order to map \hat{F}^1 into the smart camera heads.

4.3. Deeper optimization : Tuning the kernel size of the first convolutional layer

The other factor of degradation given by Equation (6) is k_i . As the hardware cost for each kernel size grows in K^2 , we retrain the minimal network $\hat{H}^M \circ \hat{F}^1$

⁹Results reported by the Intel Quartus 19.3 software using 8-bit-inputs multipliers.

390 with different kernel size for the front-end layer. We keep the nominal $K_1 = 11$
 as a reference point and report the training results for $K_1 \in [3; 5; 7; 9]$. How-
 ever, the size of the features (Equation 9) at the output of this layer should be
 maintained constant across all the experiments in a way that the total amount
 of computations for the back-end \hat{H} remains the same for all configurations of
 395 \hat{F}^1 . Maintaining a constant back-end computational workload also requires to
 decrease the input resolution to $W_{in} \in [219; 221; 223; 225]$, as shown in equa-
 tion 9 where p and s respectively stand for the padding and stride parameters
 of the convolution layer.

$$W_{out} = \frac{W_{in} + 2 \times p - K}{s} + 1 \quad (9)$$

400 Figure 12(a) displays the performance variations due to kernel size shrinking.
 This time, the dataflow required resources of the layer F^1 with a kernel size of
 $K_1 = 3$ and $K_1 = 5$ are within the range of a smart camera capacity based on a
 Cyclone III FPGA, or $K_1 = 7$ for a more recent technology like Cyclone 10GX
 devices.

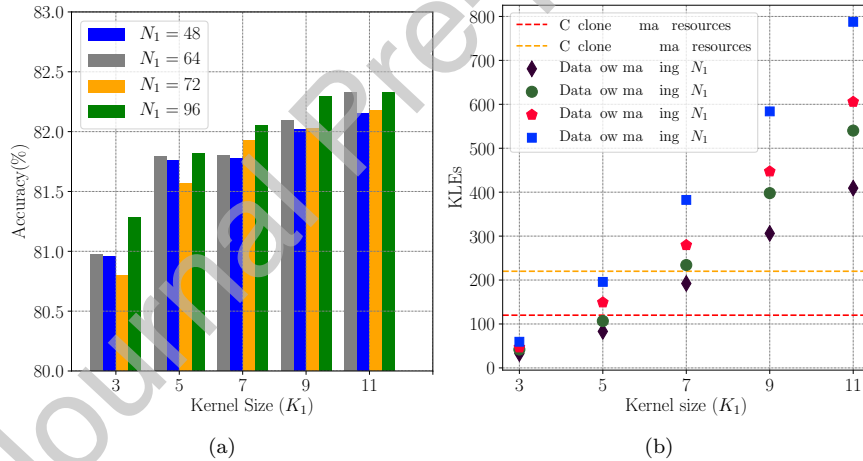


Figure 12: (a) Kernel size impact on the accuracy of the minimalist network $\hat{H}(N = [128, 192, 192, 128])$ according to 4 feature configurations for the first layer. (b) Synthesis results of 5 kernel size for the front-end dataflow implementation.

405 Equation (10) introduces a Hardware mapping Efficiency metric, HE_{index} ,
 that highlights the most efficient CNN in terms of logic utilization and accu-
 racy compared to the reference AlexNet architecture ($FC \circ H^M \circ F^1(N_1 = 96, K_1 = 11)$). The numerator represents the hardware cost difference between
 the reference AlexNet network and the tuned networks whereas the denomina-
 tor leverages the difference of accuracy, for the first layer mapping. With this
 410 metric, more hardware saves and small accuracy drops translate into a high

415 HE_{index} , as shown in Figure 13. This figure shows that the $(N_1 = 48, K_1 = 5)$ setup has the highest HE_{index} , and delivers the best trade-off between accuracy losses and resource gains. In the next studies on multi-view merging, we keep all networks with the first layer \hat{F}^1 below $N_1 = 48, K_1 = 7$, namely, all configurations where the first layer can be potentially mapped into the front-end FPGA logic.

$$HE_{index} = \frac{\text{Cost}_{hw}(H^M, F^1) - \text{Cost}_{hw}(\hat{H}^M, \hat{F}^1)}{\text{Acc}(H^M, F^1) - \text{Acc}(\hat{H}^M, \hat{F}^1)} \times \frac{\text{Acc}(H^M, F^1)}{\text{Cost}_{hw}(H^M, F^1)} \quad (10)$$

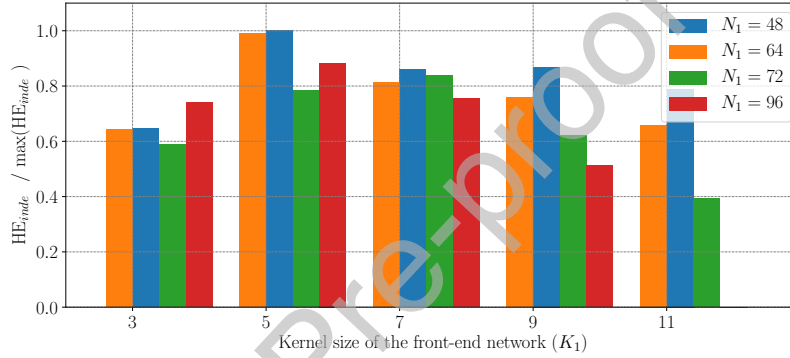


Figure 13: The hardware mapping efficiency index for all trained networks considering the first layer \hat{F}^1 . All values in the figure are normalized with the best architecture index $(N_1 = 48, K_1 = 5)$. Higher HE highlights a better trade-off between hardware Cost and accuracy loss.

4.4. Boosting CNN performance with multi-view data merging and cost analysis

420 According to the constraints described in section 3, our multi view system is composed of several smart camera heads with the unique ability to process one layer (F^1) of a CNN at sensor speed. From the results of the previous section, this kind of architecture can be implemented on relatively small FPGAs at the cost of performance degradation compared to the baseline CNN. In this section, we introduce the concept of [10] to boost the network classification accuracy to reach the same level of performance than a single view AlexNet, considered as a strong baseline. Unlike MVCNN, our system is constrained by the number of 425 hardwired multipliers embedded on the smart camera, thus the function G (i.e. view-pooling) regroups the data from each $\hat{F}^1(\theta_i)$ where \hat{F}^1 is the first layer of the CNN whereas the original MVCNN work suggests to regroup the features from the last layer of the convolution section of AlexNet. We fine-tuned the previous single view network with 2 to 6 views as presented in the figure 14(top). 430 We keep only the first layer dataflow architectures for which the network accuracy drop is less than 0.5% to discuss about the efficiency of the multi-view

system on the ModelNet40 dataset. In equation 11, we introduce a simple metric, the multi-view normalized hardware efficiency index, to outline the most efficient \hat{F}^1 architecture in respect to the number of views needed to recover from previous damages done to the original network. A low number of camera with a high count of logic resources difference from the reference architecture F^1 will give the best efficiency index. There are 3 configurations that potentially fits a smart camera based on a Cyclone III 120KLEs to reach out the AlexNet accuracy baseline(83%) with only 2 views. The least resource expensive layer $\hat{F}^1(N_1 = 48, K_1 = 3)$ requires 4 cameras to recover from degradation. The most efficient solution calls for 2 cameras at the cost of a larger kernel size ($K_1 = 5$) but the number of features remains the same.

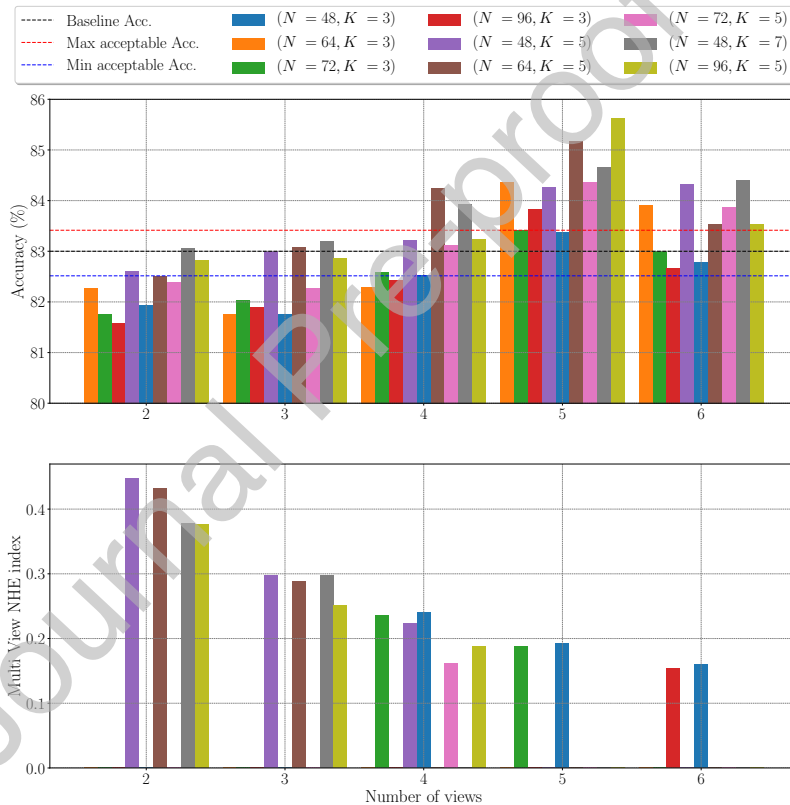


Figure 14: The upper figure shows the multi-view accuracy of all 9 setups from 2 to 6 views. The lower and upper bounds represent the limits at which we consider the system to be equivalent to the original AlexNet network. At the bottom, the multi-view network hardware efficiency (at equivalent accuracy) highlights the setups with the lowest resource cost and the lowest number of camera. (The higher the better)

$$\text{MVNHE}|_{\text{Acc} \approx \text{Acc}_{\text{AlexNet}}} = \frac{\text{Cost}_{\text{hw}}(H^M, F^1) - \text{Cost}_{\text{hw}}(\hat{H}^M, \hat{F}^1)}{\text{Cost}_{\text{hw}}(H^M, F^1) \times V} \quad (11)$$

5. Experimental Results on Camera Performances and discussion on the back-end system

We chose 2 front-end FPGAs to map 2 different layer configurations:

Cfg1 ($N_1 = 48, K_1 = 3$) and **Cfg2** ($N_1 = 48, K_1 = 5$). The first one is a Cyclone III with 120 KLEs using **Cfg1** while the second is a Cyclone 10GX with 220 KLEs for the other configuration. Both configurations are translated from the Pytorch model to VHDL using the DELiRium DHM tool¹⁰

All convolutions are directly mapped into $N_1 \times K_1 \times K_1$ 8 bits hardware multipliers while the weights are stored in registers. The synthesis results are shown in Table 2. As a result of previous optimization, the layer with ($N_1 = 48, K_1 = 3$) fits the target FPGA and requires 59% of logic elements for the 4 cameras setup using **Cfg1** on Cyclone III FPGA. The second, **Cfg2**, requires approximately twice the resources of **Cfg1**. However, as shown in figure 11, it demands half the cost in terms of camera needed to recover from the degradation. Both setups are able to run at sensor clock rate.

Table 2: Front-end synthesis results on a low-end FPGA : The full design includes the sensor configuration state machine, the image preprocessing, the communication IP and L1 on the Cyclone III based smart camera head.

(N_1, K_1)	Conv1	Full design(% Total)	Bandwidth MB/s (30 img/s)
(48,3)	70806 LEs + 28740 Registers	59% (Cyclone III)	1.001
(48,5)	146180 LEs + 113920 Registers	109% (Cyclone III)	1.001
(48,5)	39561 ALMs + 112681 Registers	69% (Cyclone 10GX)	1.001

The resulting smart camera is able to accelerate the first layer of this CNN in real time (i.e 30 fps) with a sensor base clock at 53MHz. Images of the ModelNet40 dataset are injected into the cameras heads for testing. Each line coming from the max-pooling layer (27x48 bytes) are formatted into 27 Ethernet frames and sent to the back-end. At 30 images per second, the communication consumes 1MBytes/sec of the available gigabit Ethernet bandwidth, which is far below the capacity of the theoretical Giga Ethernet bandwidth (125MBytes/sec). For sake of study completeness, the second to fifth layers and the classifier inference are ported on three different embedded platform's for our experiments. First, we choose a Jetson Nano development boards in CPU and GPU mode because it supports a straight forward replication of the training and evaluation procedures with native supports of the Ubuntu OS and Pytorch frameworks. The last platform is an Intel Cyclone V SoC 77 KLEs programmed with an OpenCL

¹⁰<http://sma-rty.com/delirium-2/>

based accelerator (PipeCNN [48]) to reflect at most the real performance of our future complete prototype of multi-view smart camera.

475 Table 3 summarizes the inference time of the different setups. As the number of MAC operations of the custom CNN back-end \hat{H} is much lower (0.205 GMACs) than the original AlexNet (0.66 GMACs), the processing time is $2\times$ faster on both CPU and GPU mode. More interestingly, it is 75% faster with PipeCNN on a Cyclone V SoC FPGA where the first configuration **Cfg1** 480 catches 4% of the total computational time. This must be put in perspective with the actual inference time of the whole AlexNet CNN at equivalent accuracy. In the second context **Cfg2**, the total time to compute the whole CNN is approximately the same than **Cfg1** but the first layer catches 9% of the processing time. Remind that we compare our solution against the conventional AlexNet CNN which first layer in PipeCNN is the second most time intensive 485 sequence (69.39 ms) behind the layer 2 (108.34 ms). In a full FPGA accelerator context and considering 4 standard cameras, the system should record all input images into its memory ($4 \times 3 \times 219 \times 219$ bytes) and stores them into the accelerator memory before the sequence of operations starts. Conversely, our 490 system has already processed the first layer at no additional time cost but the pipeline latency to output the first data. After receiving a line of the feature map from the 4 smart cameras, the central FPGA accelerator gathers the data in real time via a VHDL description of the function $G(\text{view-pooling})$ right after the data link, and calls for no more than four 8-bits comparators in a 4 view 495 setup. Finally, the resulting feature of size $48 \times 27 \times 27$ bytes (34 Kbytes) can be stored into the external memory or can be directly processed by the second layer.

Table 3: Inference time on ARM Cortex A57, embedded GPU, FPGA (PipeCNN) and Intel CPU when running the MVCNN optimized for the target camera.

Device	ARM Cortex-A57 1.43 GHz	Jetson Nano MaxN 0.92 GHz	Cyclone V SoC PipeCNN 150 MHz
AlexNet CNN	221.6 ms	31.5 ms	367.5 ms
$(48, 3, \hat{H})$ CNN(all layers)	119.5 ms (-46%)	13.7 ms (-56%)	91.3 ms (-75%)
Cfg1 CNN(w/o 1st layer)	106.2 ms (-52%)	0.5 ms	88.1 ms (-76%)
$(48, 5, \hat{H})$ CNN(all layers)	120.1 ms (-46%)	13.8 ms	96.0 ms (-73%)
Cfg2 CNN(w/o 1st layer)	104.7 ms (-52%)	0.6 ms	87.6 ms (-76%)

6. Conclusion and Future Works

500 This paper has detailed the design of a multi-view smart camera based on FPGAs and efficiently distributing the inference of a CNN with a near-sensor multi-FPGA accelerator architecture. We demonstrated that it is possible to

reduce the number of AlexNet parameters with a recognition rate equivalent to a baseline AlexNet network on the ModelNet40 dataset. By strongly optimizing the network, the first layer is fully ported to the camera heads, configuring each head to output new convolution results at each clock cycle of the image sensor. The head output data is transferred to a GPU or an FPGA that carries out the inference of the optimized network. Experimental results show that the proposed design method can produce a network with similar accuracy to a baseline single-view AlexNet with a number of convolutions divided by a factor of 3 at the price of inserting 4 camera heads in the system. A future work will consist in studying the benefits of this algorithm and architecture co-design method on deep networks consuming natively very few MAC operations such as MobileNetV1 [8], MobilenetV2 [49] or ShuffleNet [50] in order to embed even more layers on the camera heads, and to increase the camera performance on both classification and processing speed. Finally, a full custom camera based on FPGA-powered head and body will be prototyped. More precisely, the central computing node will be based on a full custom FPGA accelerator perfectly synchronizing frame captures and continuing the inference up to the classification stage.

References

- [1] C. Potthast, A. Breitenmoser, F. Sha, G. S. Sukhatme, Active multi-view object recognition: A unifying view on online feature selection and view planning, *Robotics and Autonomous Systems* 84 (2016) 31–47. doi:10.1016/j.robot.2016.06.013.
URL <http://dx.doi.org/10.1016/j.robot.2016.06.013>
- [2] A. Thomas, V. Ferrari, B. Leibe, T. Tuytelaars, B. Schiele, L. Van Gool, Towards Multi-View Object Class Detection doi:10.1109/CVPR.2006.311.
- [3] H. P. Chiu, L. P. Kaelbling, T. Lozano-Pérez, Virtual training for multi-view object class recognition, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* doi:10.1109/CVPR.2007.383044.
- [4] A. Krizhevsky, I. Sutskever, G. E. Hinton, 1 ImageNet Classification with Deep Convolutional Neural Networks, *Advances In Neural Information Processing Systems* doi:<http://dx.doi.org/10.1016/j.protcy.2014.09.007>.
- [5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 07-12-June (2015)* 1–9. doi:10.1109/CVPR.2015.7298594.
- [6] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409 (2014)* 1–14. doi:10.

- 1016/j.infsof.2008.09.005.
 URL <http://arxiv.org/abs/1409.1556>
- [7] S. Wu, S. Zhong, Y. Liu, ResNet, CVPRdoi:10.1007/s11042-017-4440-4.
 545
- [8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, arXiv e-print.
 URL <http://arxiv.org/abs/1704.04861>
- [9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, K. Keutzer, SqueezeNet: AlexNet accuracy with 50x fewer parameters and 0.5MB Model Size, arXiv e-print arXiv:1602 (2016) 1–5.
 550 URL <http://arxiv.org/abs/1602.07360>
- [10] H. Su, S. Maji, E. Kalogerakis, E. Learned-miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, Ieee Iccv (2015) 945–953doi: 10.1109/ICCV.2015.114.
 555 URL <http://vis-www.cs.umass.edu/mvcnn/docs/su15mvcnn.pdf>
- [11] Y. Feng, Z. Zhang, X. Zhao, R. Ji, Y. Gao, GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition, Cvpr (2018) 264–272doi:10.1109/CVPR.2018.00035.
 560 URL http://openaccess.thecvf.com/content_cvpr_2018/papers/Feng_GVCNN_Group-View_Convolutional_CVPR_2018_paper.pdf
- [12] B. Shi, S. Member, S. Bai, S. Member, DeepPano : Deep Panoramic Representation for 3-D Shape Recognition 22 (12) (2015) 2339–2343.
- [13] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, L. J. Guibas, Volumetric and Multi-View CNNs for Object Classification on 3D Data (2016) 5648–5656doi:10.1109/CVPR.2016.609.
 565 URL <http://arxiv.org/abs/1604.03265>
- [14] H. You, PVNet : A Joint Convolutional Network of Point Cloud and Multi-View for 3D Shape Recognition.
 570
- [15] A. Kanazaki, Y. Matsushita, Y. Nishida, RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpointsdoi:10.1109/CVPR.2018.00526.
 URL <http://arxiv.org/abs/1603.06208>
- [16] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, D. Burger, A Configurable Cloud-Scale DNN Processor for Real-Time AI, 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA) (2018) 1–14doi:10.1109/ISCA.2018.00012.
 575 URL <https://ieeexplore.ieee.org/document/8416814/>
 580

- [17] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning Convolutional Neural Networks for Resource Efficient Learning, arXiv preprintdoi: 10.1051/0004-6361/201527329.
 585 URL <http://arxiv.org/abs/1611.06440>
<http://www.arxiv.org/pdf/1611.06440.pdf>
<https://arxiv.org/abs/1611.06440>
- [18] A. ProstBoucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, V. Leroy, Scalable High-Performance Architecture for Convolutional Ternary Neural Networks on FPGA, in: Proceedings of the International Conference on
 590 Field Programmable Logic and Applications - FPL '17, 2017.
 URL <https://hal.archives-ouvertes.fr/hal-01563763/>
- [19] Y. He, M. Peemen, L. Waeijen, E. Diken, M. Fiumara, G. Rauwerda, H. Corporaal, T. Geng, A Configurable SIMD Architecture with Explicit Datapath for Intelligent Learning, in: Proceedings of the International
 595 conference on embedded computer systems: architectures, modeling and simulation - SAMOS '16, 2016.
 URL http://samos-conference.com/Resources_Samos_Websites/Proceedings_Repository_SAMOS/2016/Files/Paper_19.pdf
- [20] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner,
 600 D. Neto, J. Wong, P. Yiannacouras, D. P. Singh, From OpenCL to high-performance hardware on FPGAs, in: Proceedings of the International Conference on Field Programmable Logic and Applications - FPL '16, IEEE, 2012, pp. 531–534. doi:10.1109/FPL.2012.6339272.
 URL <http://ieeexplore.ieee.org/document/6339272/>
- [21] J. Bottleson, S. Kim, J. Andrews, P. Bindu, D. N. Murthy, J. Jin, ClCaffe:
 605 OpenCL accelerated caffe for convolutional neural networks, in: Proceedings of the IEEE International Parallel and Distributed Processing Symposium - IPDPS'16, 2016, pp. 50–57. doi:10.1109/IPDPSW.2016.182.
- [22] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, G. R. Chiu, An
 610 OpenCL(TM) Deep Learning Accelerator on Arria 10, in: ACM (Ed.), Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17, ACM, Monterey, California, USA, 2017, pp. 55–64. doi:10.1145/3020078.3021738.
 URL <http://arxiv.org/abs/1701.03534>
- [23] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, J. Cong,
 615 Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs, Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC '17 (2017) 1–6doi:10.1145/3061639.3062207.
 620 URL <http://dl.acm.org/citation.cfm?doid=3061639.3062207>
- [24] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, F. Berry, Tactics to Directly Map CNN Graphs on Embedded FPGAs, IEEE Embedded Systems Lettersdoi:10.1109/LES.2017.2743247.

- [25] L. Li, C. Sau, T. Fanni, J. Li, T. Viitanen, F. Christophe, F. Palumbo, L. Raffo, H. Huttunen, J. Takala, S. S. Bhattacharyya, An integrated hardware/software design methodology for signal processing systems, *Journal of Systems Architecture* 93 (2019) 1–19. doi:10.1016/j.sysarc.2018.12.010.
- [26] Z. Wu, S. Song, 3D ShapeNets : A Deep Representation for Volumetric Shapes.
- [27] D. Maturana, S. Scherer, VoxNet : A 3D Convolutional Neural Network for Real-Time Object Recognition (2015) 922–928.
- [28] A. Canziani, A. Paszke, E. Culurciello, An Analysis of Deep Neural Network Models for Practical Applications, arXiv e-print.
URL <http://arxiv.org/abs/1605.07678>
- [29] K. Abdelouahab, M. Pelcat, F. Berry, J. Sérot, Accelerating CNN inference on FPGAs: A Survey, Tech. rep., Université Clermont Auvergne (2018).
URL <https://hal.archives-ouvertes.fr/view/index/docid/1731136>
- [30] S. I. Venieris, A. Kouris, C.-S. Bouganis, Toolflows for Mapping Convolutional Neural Networks on FPGAs, *ACM Computing Surveys* 51 (3) (2018) 1–39. doi:10.1145/3186332.
URL <http://dl.acm.org/citation.cfm?doid=3212709.3186332>
- [31] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, in: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*, FPGA, 2015, pp. 161–170. doi:10.1145/2684746.2689060.
URL <http://dl.acm.org/citation.cfm?id=2689060>
- [32] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, H. Yang, Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, in: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*, ACM, New York, NY, USA, 2016, pp. 26–35. doi:10.1145/2847263.2847265.
URL <http://doi.acm.org/10.1145/2847263.2847265>
- [33] V. Sze, Y.-H. Chen, T.-J. Yang, J. Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, *Proceedings of the IEEE* 105 (12) (2017) 2295–2329. doi:10.1109/JPROC.2017.2761740.
URL <http://ieeexplore.ieee.org/document/8114708/>
- [34] S. Venieris, C. Bouganis, FpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs, in: *Proceedings of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines - FCCM '16*, 2016, pp. 40–47. doi:10.1109/FCCM.2016.22.

- [35] S. Venieris, C. Bouganis, Latency-Driven Design for FPGA-based Convolutional Neural Networks, in: Proceedings of the International Conference on Field Programmable Logic and Applications - FPL '17, 2017.
- [36] H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, in: Proceedings of the IEEE International Conference on Computer Vision - ICCV '15, IEEE, 2015, pp. 945–953. doi:10.1109/ICCV.2015.114.
URL <http://ieeexplore.ieee.org/document/7410471/>
- [37] K. Abdelouahab, M. Pelcat, F. Berry, The Challenge of Multi-Operand Adders in CNNs on FPGAs: How not to solve it!, in: Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation - SAMOS'18, 2018.
URL <http://arxiv.org/abs/1807.00217>
- [38] E. A. Lee, T. M. Parks, Dataflow Process Networks, in: Proceedings of the IEEE, Vol. 83, 1995, pp. 773–801. doi:10.1109/5.381846.
- [39] R. Banner, I. Hubara, E. Hoffer, D. Soudry, Scalable methods for 8-bit training of neural networks, Advances in Neural Information Processing Systems 2018-Decem (2018) 5145–5153.
- [40] H. Naganuma, R. Yokota, Accelerating Convolutional Neural Networks Using Low Precision Arithmetic, Sighpc.Ipsj.or.Jp 1–3.
URL <http://sighpc.ipsj.or.jp/HPCAsia2018/poster/post108s2-file1.pdf>
- [41] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, L. Chang, Compensated-DNN: Energy Efficient Low-Precision Deep Neural Networks by Compensating Quantization Errors 1–6.
- [42] D. Kim, H. Y. Yim, S. Ha, C. Lee, I. Kang, Convolutional Neural Network Quantization using Generalized Gamma Distribution doi:arXiv:1810.13329v1.
URL <http://arxiv.org/abs/1810.13329>
- [43] J.-H. Luo, J. Wu, An Entropy-based Pruning Method for CNN Compression.
URL <http://arxiv.org/abs/1706.05791>
- [44] Y. Cheng, D. Wang, P. Zhou, T. Zhang, A Survey of Model Compression and Acceleration for Deep Neural Networks, arXiv preprint.
URL <http://arxiv.org/abs/1710.09282>
- [45] B. T. Phong, Illumination for Computer Generated Pictures, Communications of the ACM 18 (6) (1975) 311–317. doi:10.1145/360825.360839.
- [46] A. Appel, Some techniques for shading machine renderings of solids (1968) 37 doi:10.1145/1468075.1468082.

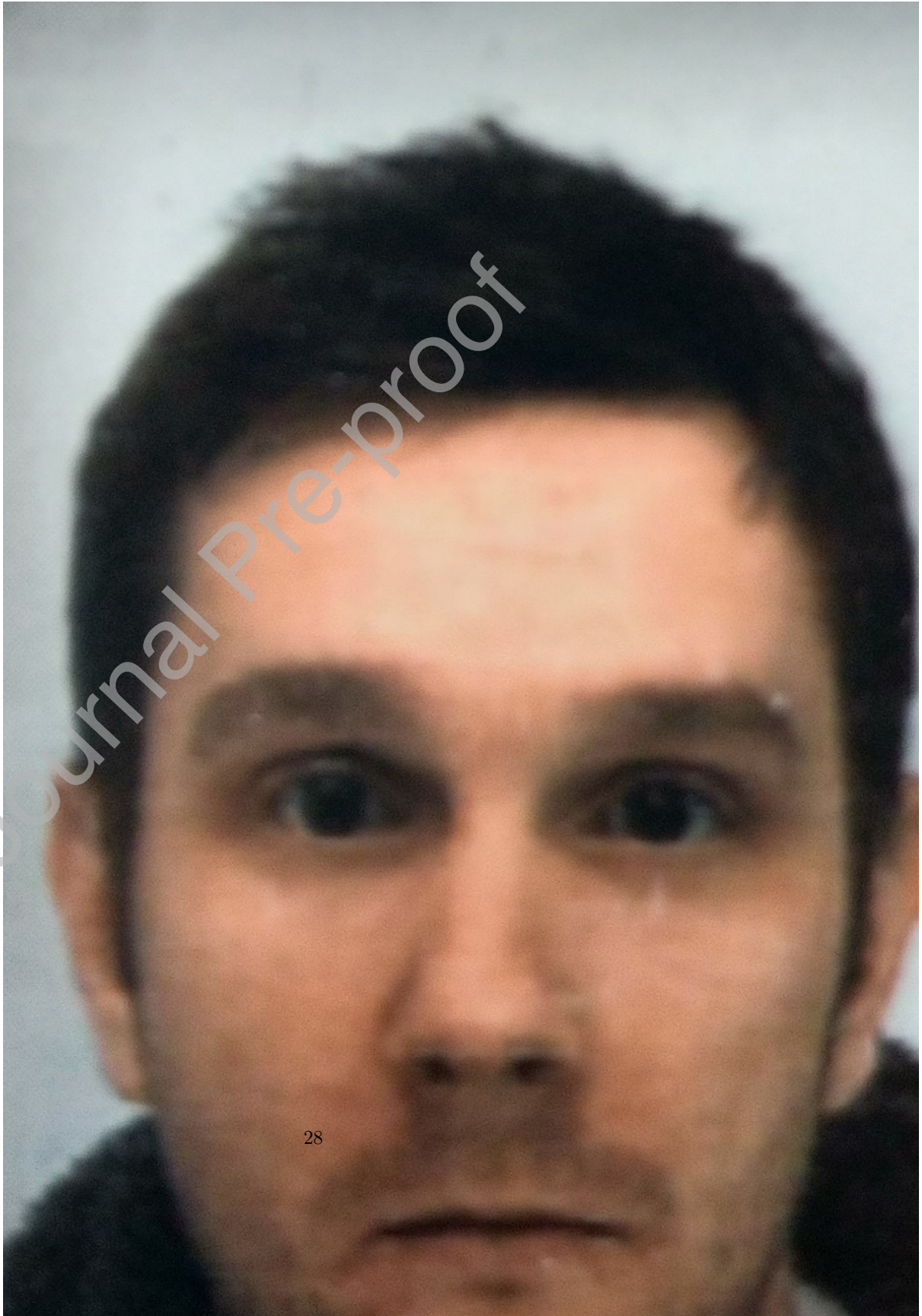
- [47] B. Steiner, Z. Devito, S. Chintala, S. Gross, A. Paszke, F. Massa, A. Lerer, G. Chanan, Z. Lin, E. Yang, A. Desmaison, A. Tejani, A. Kopf, J. Bradbury, L. Antiga, M. Raison, N. Gimelshein, S. Chilamkurthy, T. Killeen, L. Fang, J. Bai, PyTorch: An Imperative Style, High-Performance Deep Learning Library, NeuroIPS (NeurIPS).
705
- [48] D. Wang, PipeCNN: An OpenCL-based FPGA Accelerator for Convolutional Neural Networks, in: Proceedings of the International Conference on Field-Programmable Technology - FPT '17, 2017.
710 URL <https://github.com/doonny/PipeCNN>
- [49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks.
URL <http://arxiv.org/abs/1801.04381>
- [50] N. Ma, X. Zhang, H. T. Zheng, J. Sun, Shufflenet V2: Practical guidelines for efficient cnn architecture design, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11218 LNCS (2018) 122–138. doi: 10.1007/978-3-030-01264-9{_}8.
715

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.

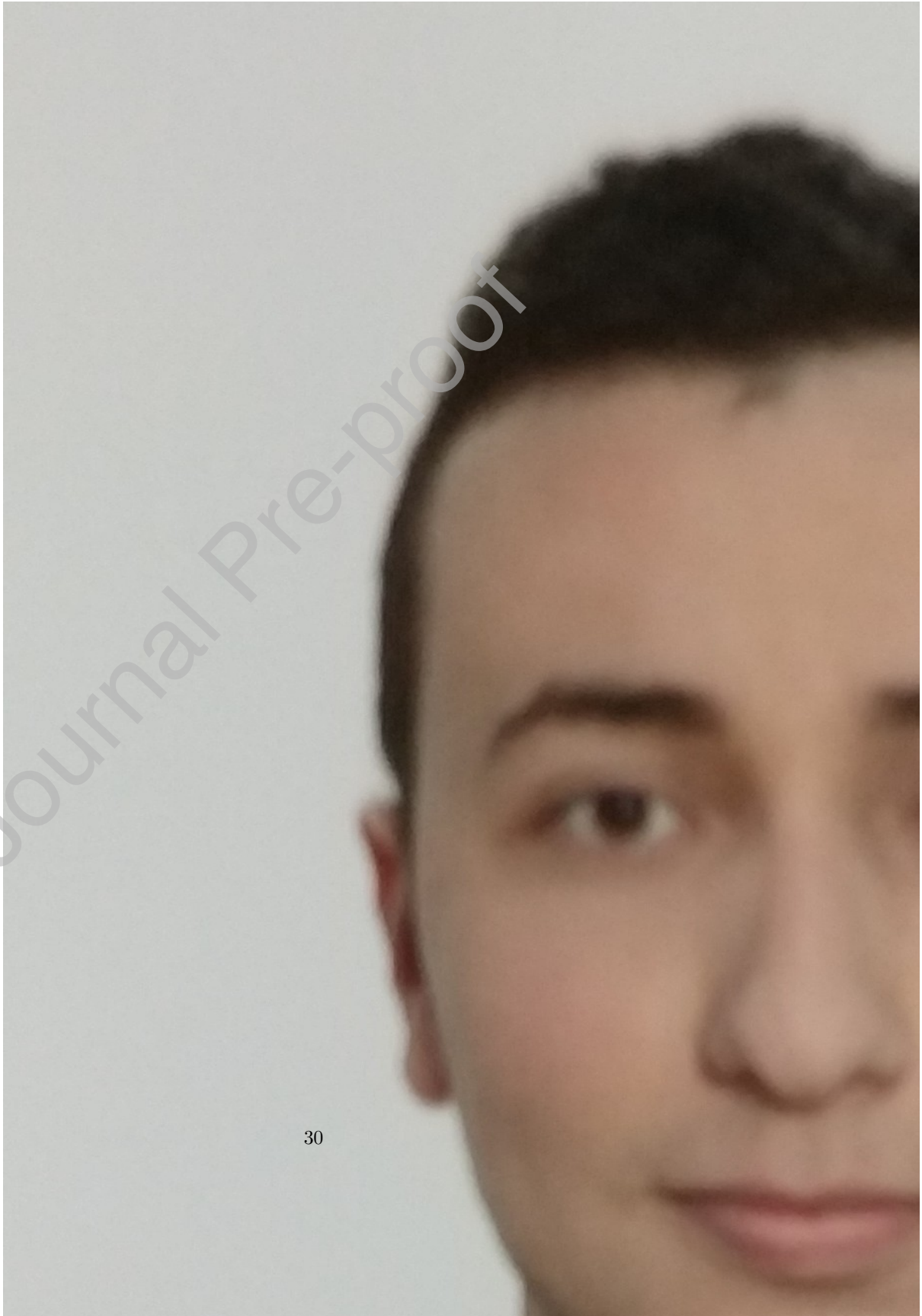
The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Journal Pre-proof



Jonathan Bonnard received the title of embedded system architect at the Université Pierre et Marie Curie in 2008, Paris, France. After a 9-year experience in FPGA and ASIC circuit design at the laboratoire de physique corpusculaire in Clermont Ferrand (LPC), he started a thesis at the Université Clermont Auvergne in 2017 under the supervision of Pr. François Berry and Dr. Maxime Pelcat. His research has focused on the design of a multi-view smart camera system based on FPGAs.

Journal Pre-proof



Kamel Abdelouahab is a research and development engineer at Sma-RTy SAS. He obtained his Ph.D in electronics and system architectures in 2019 from Universite Clermont Auvergne (Clermont-Ferrand). He also received a master degree of engineering in electronics from Ecole Nationale Polytechnique (Algiers) in 2015. His research interests are related to embedded deep learning, smart cameras, FPGA-based computer vision systems.



François Berry is full professor at the University of Clermont-Auvergne. He received his doctoral degree and “Habilitation” to supervise doctoral research in Electrical Engineering from the University of Blaise Pascal in 1999 and 2011, respectively. His PhD was on visual servoing and robotics at the Institut Pascal in Clermont-Ferrand. Since September 1999, he is member of the “Image, Perception Systems and Robotics group” within the Institut Pascal-CNRS. His research is focused on smart cameras, active vision, embedded vision systems and hardware/software co-design algorithms. He is in charge of a Masters in “Embedded System for Image and Sound processing” and is the head of the DREAM (Research on Embedded Architecture and Multi-sensor) team. He has authored and co-authored more than 60 journal, conference and workshop papers. He has also led several research projects (Robea, ANR, Euripides) and has served as a reviewer and a program committee member. He was co-founder of the Workshop on Architecture of Smart Camera (WASC); the Scabot (Workshop in conjunction with IEEE IROS) and also the startup WISP.



Maxime Pelcat is an Associate Professor at the INSA in Rennes. He holds a joint research appointment at IETR in Rennes and at Institut Pascal in Clermont Ferrand, two CNRS research units. Maxime Pelcat obtained his Ph.D. in signal processing from INSA Rennes in 2010, thesis resulting from a collaboration of Texas Instruments, Nice and INSA Rennes. Previously, after one year in the Audio and Multimedia department at Fraunhofer Institute IIS in Erlangen, Germany, he worked as a contractor at France Telecom Research and Development until 2006. He is an author of 50+ peer reviewed publications since 2009 in the domains of models of computation, energy efficiency, multimedia and telecommunication processing, and programming of parallel embedded systems. Maxime Pelcat has served as Guest Editor for the Springer Journal of Signal Processing Systems and is an author of the best paper award at DASIP 2014, the best demo awards at ICME 2015 and EDERC 2014 and of the book – Physical Layer Multi-Core Prototyping’ Springer, 2012