Embedded Image Processing the European Way: A new platform for the future automotive market

Tim Hotfilter, Fabian Kempf and Jürgen Becker
Institute for Information Processing Technologies
Karlsruhe Institute for Technology
Karlsruhe, Germany
hotfilter@kit.edu

Dominik Reinhardt
Bayrische Motorenwerke AG
Munich, Germany

Imen Baili
Menta eFPGA S.A.S
Valbone, France

***Abstract*—Within the European Processor Initiative (EPI) an objective is build an embedded High-Performance processing platform for future automotive applications such as autonomous driving. An embedded Field-Programmable-Gate-Array (eFPGA) enables the platform to be extended for future needs and requirements by various stakeholders. In this paper we give an overview about the project and our contributions to define the architecture of the eFPGA, which is suitable for the automotive market.**

Therefore, we describe our concept to explore the eFPGA architecture. It is motivated by a sound use case that deals with face recognition based on current neural networks. During the scope of the work we describe how the application is carefully mapped on the different domains of the EPI platform to make it more safe and secure as well as performant. As a result, we will find an apt eFPGA configuration, which can host common but also future neural network applications and a mapping of common image processing tasks.

***Index Terms*—EPI, eFPGA, neural networks, image processing, face recognition, CNN accelerator, automotive safety and security**

I. Introduction

Recent developments and publications indicate a clear trend towards more application specific processors. Very prominent examples are machine learning and data processing applications. In the past, those applications were executed on High Performance Computers (HPC) located in data centers or on warehouse computers. To reduce the energy consumption and to accelerate the computation of those algorithms developers have started to deploy them on more and more powerful Graphic Processing Unit (GPUs) in which the parallel data processing structures can be exploited in an efficient way. Therefore, GPUs found their way into HPC data centers. Highly parallel algorithms are very common in the domains of artificial intelligence, like recent deep neuronal networks (DNNs), and computer vision. This trend empowered the development of novel heterogeneous compute architectures, which are also installed in HPC datacenters. For instance, Field Programmable Gate Arrays (FPGAs) and specified AI-Accelerators are deployed to increase the overall performance. Due to Moore's law, more and more functionality and IP can be integrated into a single chip. This leads to a monolithic integration of CPUs, GPUs, accelerators and even FPGAs on a single die or in a single package. For instance, Xilinx's Zynq tightly couples ARM cores and a FPGA for user logic. Moreover, they recently released the adaptive compute acceleration platform (ACAP), which introduces dedicated hardware accelerators for DNNs.

The European Processor Initiative (EPI) aims to develop a heterogeneous multi-core processor consisting of ARM CPUs, hardware accelerators and an embedded FPGA. The processor and platform provided by the EPI consortium will target a wide range of applications, form high performance exascale computing to embedded platforms for autonomous driving.

Autonomous driving tasks are a good example for a computationally high demanding use case. Path planning, decision-making and perception require a huge amount of sensor data to be processed. As of today, they exceed the computational power of currently available automotive processors, like Infineon AURIX TC3xx or Renesas H-Series. Therefore, EPI targets to develop an embedded High Performance Computer (eHPC) suitable for autonomous driving.

Future autonomous driving sub-tasks are very likely to be realizes as a service-oriented computing architecture. Accordingly, the authors in [1] define different levels of service-oriented architectures. The lowest level is referred to as a Software-as-a-service (SaaS), which is defined by functionality and information on-demand. In this paper, we will show how to exploit the eFPGA on the EPI chip for such a software-as-a-service based on an automotive use case. Therefore, the paper is structured in the following way: First, we will give a short background about the project and context. Afterwards we describe the current issues that have to be solved. In the next section, we propose a convolutional neural network (CNN) accelerator architecture, which addresses compute-intense parts of future automotive applications. In the end, we give a short conclusion and summarize our findings.

In addition to the software as a service, our use case follows another purpose. While the function of the eFPGA is reconfigurable at run-time, the actual chip layout has to be defined in advance. In particular, we have to define the organization and the amount of look-up tables, Flip-Flops and macro blocks, like digital signal processors (DSP) or multiply accumulate units (MAC). Moreover, the overall area of the eFPGA in the chip is constrained and has to be utilized in the best way possible. The sound use case helps us to define an

eFPGA chip layout, which is suitable for common and future automotive needs.

In summary our contributions are threefold:

- We present our concept which enables us to achieve a manifold eFPGA configuration for the future needs and requirements.
- We show a scalable architecture design for the eFPGA for architecture exploration, which implements a convolutional neural network (CNN).
- We describe a face recognition use case, which backs our architecture and can be used as scenario for further evaluation.

II. Background

A. A quick introduction to CNNs

A CNN is a sub-type of a DNN [2]. In the same way it consists of layers, but adds convolutional layers. They are sequentially connected with each other, followed by a little number of fully connected layers to classify the features generated by the convolution layers. Each convolutional layer takes the features from the previous layer or from the input and applies a number of convolution kernels in the spatial dimensions by basically being moved around on the input data. The values of the convolution kernels are also called weights, which are determined during the training phase. The way in which spatial information are taken into account, benefits image processing tasks widely, which makes CNNs very popular when images or videos have to be processed.

Similar to other neural networks a non-linear activation function is applied to the feature map after each layer. Moreover, convolution layers are followed by pooling layers. The idea is to reduce the size of the output feature map by keeping the maximum value of a certain spatial area and removing the other values. A 2x2 pooling for instance reduces the feature map by 4.

B. The EPI Platform

The overall EPI chip will be available as general purpose processor (GPP) for data center applications and an embedded version, also called embedded High-Performance Chip (eHPC) [3]. The GPP is organized in four domains: At the heart of the chip are ARM general-purpose processors with scalable vector extensions. High-performance applications are supported by the EPI-accelerator cores based on the popular RISC-V ISA with extensions for vector calculations and deep learning. In addition, the GPP platform offers an MPPA tile by Kalray in order to support highly distributed tasks. To support future applications and customization the GPP is equipped with an embedded FPGA (eFPGA) provided by Menta. It allows for dedicated accelerators based on the customers' needs. Each tile is connected via a high-bandwidth network-on-chip (NoC).

The eHPC chip extends a GPP with a safety processor, which is based on the most-recent AURIX series provided by Infineon. It acts as a safety island and sits between the GPP and the peripherals. This additional certified processor can guard and supervise in- and out-going signals, which makes the eHPC chip suitable for the future automotive needs like autonomous driving from the signal processing, decision planning to control of the actuators.

III. Related work

Hardware accelerators for neural networks are not an entirely new topic. First attempts on FPGAs were already made in the 90s [4]. With the rise of more complex and cheaper ASICs, the development of architectures tailored for CNN gained pace very quickly. The first powerful architectures came up in the early 2010s [5].

As of today most architectures in both production and research are based on systolic arrays consisting of small MAC units. A well-known architecture is the 2017 presented Tensor Processing Unit (TPU) by Google [6]. However, many other accelerators came up in the same period. Each addresses different yet unsolved bottlenecks, like memory bandwidth or a lack of computation power in different ways. The Eyeriss architecture by Chen et al. [7] features a reconfigurable systolic array that supports multi-cast to many processing elements as well as a very efficient input data pruning strategy, which skips zero values on the fly. Another interesting approach is presented by Lee et al. [8]. They offer a fully configurable number precision from 1 to 16 bit. Moreover, they try to address the memory bottlenecks with a cache-like Aligned Feature Loader (AFL).

In contrast to the systolic arrays, Qiu et al. [9] presented an approach in which they do not rely on a hardware structure for matrix multiplications, but on rather small accelerators for a single convolutional operation, consisting of 3x3 multipliers and an adder tree. The advantage of the architecture is a low latency on single input data, as data can be processed in a single shot without grouping multiple input as batches. Moreover, they found and proved the fact that convolution layers in neural network are in general compute-centric and fully connected layers memory-centric. This means that convolution layers are accountable for the largest share in computation time and rather less memory content has to be exchanged during execution. On the other side, fully connected layers require much more memory bandwidth in order to run in an efficient way.

Face detection algorithms date back to the early 90s when Turk et al. presented their EigenFaces algorithm [10]. While this holistic approach extracts facial features from the input images in a pixel-wise fashion, recent work in this area is mainly based on deep neural networks. This trend started with DeepFaces by Taigman et al. [11]. They showed in 2014 based on the Labeled Faces in the Wild dataset, a comparable accuracy (97.35%) to humans (97.53%) [12]. As shown in a survey [13] of face recognition current successful approaches split the algorithm in three parts. First faces have to be detected in order to be aligned and centered. In the second step a face pre-processing tries to reduce the impact to different illumination or perspective angle. Only the third step deals with the extraction of facial features, which can be used to match the faces to the according person.

Fig. 1. Exemplary face detection

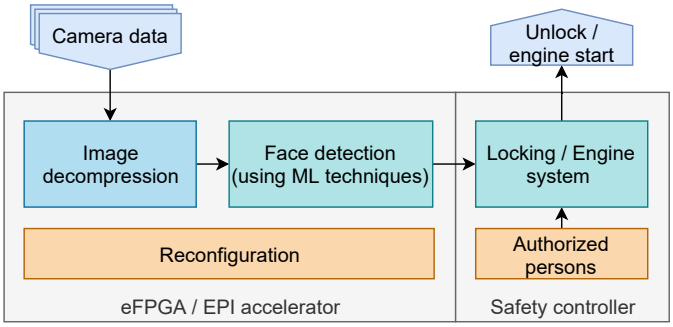

Fig. 2. Overview of the face recognition use case

IV. Use case

The process to find a well-fitting eFPGA architecture presented in our work is backed by a current application in accordance with the automotive stakeholders. Face detection gained popularity in many areas in the recent years. Famous examples are for instance smartphone unlocking or clustering of photographs. In Figure 1 you see a first approach to detect an exemplary face, marked with a blue square for visualization purposes. Our use case moves the eFPGA technology to the automotive sector and is separated into two development scenarios.

The first scenario is recording the outside environment of the car. Our use case is to unlock the car as soon as an authorized person approaches the car. It is also conceivable to lock functions within the car, like avoid starting the engine, if a non-eligible person takes place on the driver seat. An example use case might be that your child is able to unlock the car and take place inside; however, you want to prevent him or her from starting the engine. For this achievement, the cameras to create a surround-view of the car are used. Typically, this vehicle functionality is used to facilitate the parking of the car and prevent damages by visualizing the full environment around the vehicle for the driver and alerting close obstacles.

The second scenario takes camera input from the inside of the car, in particular the driver's seat. Our use case is to detect gestures and mimic of the driver and to estimate his mood based on this information. An example use case might be that in case of bad mood, different kinds of comforting light impressions or calming scents are used to enable stress-less driving. If the driver has to focus on a difficult driving maneuver, the car removes distracting environmental sounds or unnecessary visual signals. For this application, in-car cameras are used to detect drowsiness and gestures. Typically, these vehicle functionalities are already common features within the car and given hardware can be reused.

V. Conceptualization

As described before, our use case for the eFPGA copes with camera-based face recognition. A broad overview of the concept is given in Figure 2. Our approach comprises an end-to-end solution on the EPI chip. Input data from multiple cameras around the vehicle and inside of it is passed into platform. This data is evaluated in different domains of the chip and in the end a signal is generated which unlocks the car or starts the engine.

The upstream data-decompression and the actual face detection happens in the eFPGA. A hardware implementation of the face detection allows for a more efficient computation. However, this structure has to be fixed during the chip design. The main advantage of the eFPGA is the ability to reconfigure the circuit during run-time and thus combines both advantages. With this concept it is conceivable that the face detection algorithm gets configured as soon as the car stops and the eFPGA resources can be freed when the car is in driving mode to host other applications.

The overall processing chain is split into two parts. The face detection algorithm, which runs on the eFPGA and the evaluation of the results on the safety island. This separation allows us to run the face detection in a faster way, while the interpretation of detected faces and the resulting unlock can be handled in a safe fashion. This means on the one hand the evaluation has a higher availability through lock-stepped cores thus it is protected against random glitches and failure. On the other the isolated piece of software can be designed in a secure way in order to protect the system from external threads. However, the reliability and confidence of the face detection algorithm has an impact on the overall security of the system. To address this issue, we pursue common approaches to this. For instance, the evaluation of different images from different angles increases the confidence when the face do not differ. In addition, a depth or time-of-flight camera allows to detect simple images of authorized persons.

A. Face Recognition Algorithm

The face recognition algorithm is based on state-of-the-art convolutional neural networks, which are very suitable for image processing tasks. Our neural network topology is based on the deep CNN FaceNet by Schroff et al. [14]. The architecture of this network is broadly inspired by the popular Inception-ResNet V1 network by Szegedy et al. [15]. The network is trained using the and VGGFace2 data-set [16], which features about 3.3M faces of ~9000 different people.

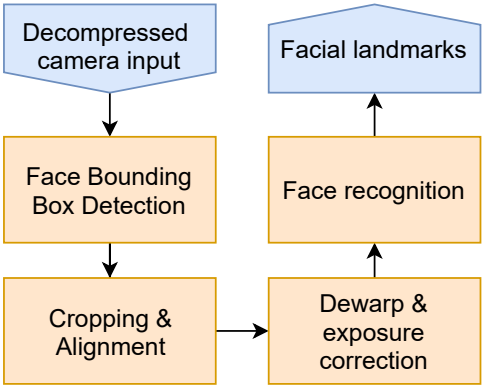

Fig. 3. Steps of the face detection algorithm

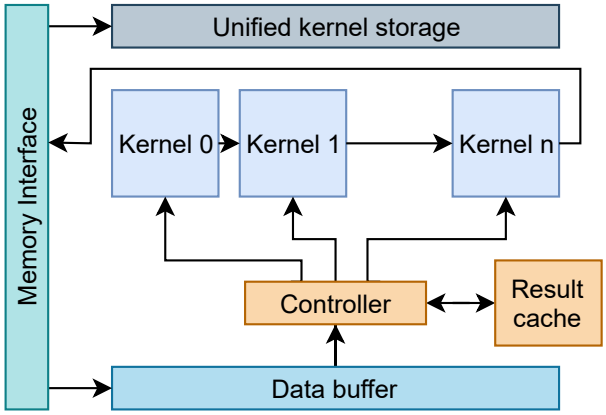

Fig. 4. Overview of our architecture

However, as described before the actual face recognition algorithm does not work well on raw camera input. As described in [14] faces have to be centered and aligned as well as dewarped in advance. The steps of the algorithm are depicted in Figure 3 Out of this reason, an upstream face detection algorithm gives the bounding boxes of faces available in the image. In a next step the input image can be cropped and aligned to comply to the expected input of the recognition network. The detection algorithm is also based on CNNs, this makes it again easy to map the algorithm on our accelerator architecture. The network topology is inspired by a work by Zhang et al. [17]. In particular, their work addresses multiple task at the same time: face classification, a bounding box detection and the localization of facial landmarks, like eyes, mouth. In our application we focus on the bounding box to crop and align the input data.

B. Neural network topology

As described before we have to use two adjacent neural networks. A face detection network aligns the input data to make the face recognition network faster and more efficient. Since we focus not on the algorithm itself but on an efficient and fast hardware implementation of the network, it is crucial in the beginning to have a close look at the network topology.

The face detection network consists of 12 layers with mostly 3x3 and some 2x2 convolution operations. The face recognition algorithm takes resized and aligned 224x224 RGB images as input. The network is based on the Incepcetion-ResNet V1 and builds mostly on 3x3 convolutions as well. Moreover, some smaller kernels like 1x3 or 1x1 and also slightly bigger such as 1x7 or 7x1 can be found. ReLUs form the non-linear activation functions in both networks. In addition, concatenate layers, poolings and normalizations have to be handled.

VI. eFPGA Architecture Overview

As mentioned in the concept chapter the images from the camera have to be processed in order to unlock the car or start the engine using face recognition. At the heart of this chain sits a convolutional neural network.

A. Basics of the architecture concept

2D-convolution operations move multiple kernels across the spatial dimensions of the input data. In each step a weighted sum of the input elements is calculated. The weights are represented in the convolution kernel. From a computational point of view, the 2D-convolution can take advantage from a high data-reuse, since some input values remain as the kernel is moved across the input data. However, the operation requires a large amount of computation effort as a multiply-accumulate (MAC) has to be derived for each input value.

To build a full CNN also other operations such as pooling and activation are necessary. However, they do not introduce a huge computational overhead, since they scale in a linear way. Especially the used ReLU activation function can be implemented using a very simple logic, which checks for the sign bit.

Many popular hardware architectures in the area of neural networks implement systolic arrays with MAC-cells, which are able to solve matrix multiplications in a very efficient way. In general, convolutions can be mapped to matrix multiplications through an unrolling process, called `im2col` [18]. However, it requires data duplication and thereby introduces latency. Moreover, the array size has to be picked very carefully to reduce the data movement and to increase the utilization. These disadvantages have less impact, when multiple input data or batches are processed at once. This happens for instance during training or in data-center applications.

With respect to our application, small accelerators for single convolutions look very promising, since we are also dealing with single input data. In addition to that, this architecture concept can be easily scaled up, by instantiating more small accelerators. This allows us to highly utilize the eFPGA structure and get a maximum performance out of the limited chip area.

B. Scaleable CNN accelerator

Based on this concept we came up with an architecture, which is depicted in Figure 4. The building blocks of the architecture and their implementation are described hereinafter.

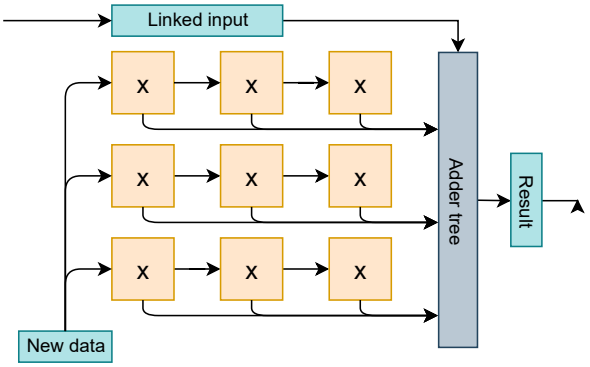

Fig. 5. Detailed architecture of the computation kernels

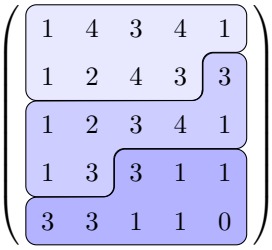

Fig. 6. Example of a larger 5x5 kernel. The computation requires three iterations.

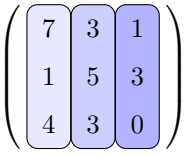

Fig. 7. Example of smaller 3x1 kernels. In this case three convolutions can be computed at once.

The architecture processes the neural network one layer after another. In the beginning the kernel weights are loaded into the unified kernel storage. This module supplies the kernel units with the weight operands. Since most convolution operations in our neural networks use 3x3 kernels, we decided for a general 3x3 kernel size (see section V-B). However, we are still able to compute larger or smaller kernel without architecture customization.

Similar to the approach by Qiu et al. incoming data is first stored into a data buffer, which functions as a line buffer. Depending on the kernel size, this structure can store up to $kernel_y - 1$ lines of the input data. This is required to realign the data, since the input image is passed line-wise into our architecture and the convolution operation needs a tiled window of the input image. The data buffer outputs input data required for the kernel each time new data arrives.

A controller block between the data buffer and the kernel, distributes the data to the available kernel units. Since images in our use case have a high correlation among each other, a small result cache is attached to the controller block. With the cached results it can decide whether the computation of a convolution operation can be skipped to save processing power. The result cache is implemented in a directly mapped way. The index is computed in the data buffer based on the nine input values. The depth of the cache is an important parameter in the design exploration step.

The computation kernels themselves have a straightforward implementation. More insights on the kernel structure are given in Figure 5. For simplicity reasons only the data flow is shown, weights are loaded into each multiplier from the unified kernel storage and are only changed when all input data is processed. Each kernel unit features nine multiply units and an adder tree, which allows us to compute one convolution each cycle. Their overall number can easily be scaled up. If more kernels have to be computed than hardware resources are available, the same image has to be processed multiple times with different weights. Although this introduces overhead we can save local memory in the unified kernel storage. Weights for the next pass-through might be prepared in background while the current kernels are processed. In addition, we added the ability to link the computation kernels as shown in Figure 4. The link allows us to compute deep convolution kernels in a simple way. For example, if the input data to the layer has a depth of 32 and 64x3x3 kernels have to be computed we can link 32 computation kernels and accumulate the result in a single step.

While the picked neural network topologies mainly use 3x3 convolution operations, they are not limited to this, but have also kernel sizes of 5x5, 1x1 or 3x1. For layers in which larger kernels are used, we split up the kernel into sub-kernels, since each calculation can be done independently from each other. The splitting of a 5x5 kernel is shown in Figure 6 in different colors for each sub-kernel. It has to be noted that we can compute a nine elements in the first batch and after that only eight elements, because we have to keep the accumulation in mind. The accumulated value replaces the values in memory and the weight of the accumulator is set to 1. If the kernel is smaller than the computation unit, we can group multiple kernels to achieve a higher utilization. An example of 3x1 kernels is depicted in Figure 7.

Apart from the convolution, operations such as activation functions, poolings and concats should not be neglected. Guo et al. describe in their survey paper [19] the distribution of operations in common CNNs. They found that only about 0.2% of the computational power is spend on these operations. Out of this reason we can map them on the ARM GPPs.

C. Further optimizations and considerations

As stated before, the objective is to have a highly efficient and low-latency execution of the face recognition algorithm. In the area of hardware accelerated neural network some optimization strategies have been established in academia and research.

Data and weight quantization is one of the most commonly applied technique. This means, that the precision of both weights and input data is reduced from 32-bit float during the training process to for instance 8-bit integers in the inference phase. It was already found in the beginning of 2010 that this has a very small impact on the overall accuracy [20]. Since integer operations can be implemented in a much simplified way and less bits require fewer resources, in addition, we can save up 20 times area and about twice the energy.

In neural network accelerators the bandwidth between the large but cheap offchip-DRAM and expensive but local SRAM

is crucial, since the according weights and input data have to be available when they are required to prevent the hardware accelerators from data starvation. An optimizing strategy, which is often applied in this context is weight and data compression. The decompression happens as close to the computation devices as possible.

To comply with our area and performance constraints, our architecture concept relies on 8-bit wide weights and input data. Moreover, compression of the input data is applied.

VII. Conclusion

In this paper, we proposed an architecture to compute convolutional neural networks under the constraints of a very limited area on an eFPGA in EPI chip. Our architecture is motivated by an at the moment very demanding automotive use-case, dealing with face recognition. Therefore, we first explored this application and its requirements to a hardware architecture. The objective of our work is to define the eFPGA layout based on the use-case. Once the eFPGA structure is build on the final chip, the basic structure is frozen but its reconfigurability remains.

During the scope of our work we present a scaleable CNN accelerator structure, which can be deployed on different eFPGA and of-the-shelf FPGA architectures. It supports state-of-the-art online optimization strategies, which makes it both powerful and efficient. Our architecture is configurable in a very flexible way, allowing tuning different design parameters ranging from the amount of kernels to the cache memory depth. With this we can determine an optimal eFPGA structure for our application, though a design space exploration.

In addition, we kept the safety aspect of our solution in mind, which is crucial to our use case but also to automotive applications in general. We selected the available partitions on the EPI platform to make the face database and comparison logic both safe and secure. The face recognition algorithm is made secure through common techniques such as multiple input image from different perspectives or an additional depth channel.

Acknowledgment

European PROCESSOR INITIATIVE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 826647

References

[1] D. Reinhardt, U. Dannebaum, M. Scheffer, and M. Traub, "High performance processor architecture for automotive large scaled integrated systems within the european processor initiative research project," in *WCX SAE World Congress Experience*. SAE International, apr 2019. [Online]. Available: https://doi.org/10.4271/2019-01-0118

[2] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, 2010, pp. 253–256.

[3] EPI, "Epi factsheet general," 2019, accessed = 2020-01-02. [Online]. Available: https://www.european-processor-initiative.eu/dissemination-material/epi-factsheet-general/

[4] M. Gschwind, V. Salapura, and O. Maischberger, "Space efficient neural net implementation," 02 1995.

[5] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 247–257, Jun. 2010. [Online]. Available: https://doi.org/10.1145/1816038.1815993

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and et al., "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[7] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

[8] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "Unpu: A 50.6tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 218–220.

[9] J. Qiu, S. Song, Y. Wang, H. Yang, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, and N. Xu, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*. Monterey, California, USA: ACM Press, 2016, pp. 26–35. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2847263.2847265

[10] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991, pMID: 23964806. [Online]. Available: https://doi.org/10.1162/jocn.1991.3.1.71

[11] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1701–1708.

[12] G. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," *Tech. rep.*, 10 2008.

[13] M. Wang and W. Deng, "Deep face recognition: A survey," *CoRR*, vol. abs/1804.06655, 2018. [Online]. Available: http://arxiv.org/abs/1804.06655

[14] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. [Online]. Available: http://arxiv.org/abs/1503.03832

[15] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[16] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *International Conference on Automatic Face and Gesture Recognition*, 2018.

[17] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, Oct 2016.

[18] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," 10 2006.

[19] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-Based Neural Network Accelerator," *arXiv:1712.08934 [cs]*, Dec. 2018, arXiv: 1712.08934. [Online]. Available: http://arxiv.org/abs/1712.08934

[20] P. W. Dally, "High-performance hardware for machine learning," Talk presented at NIPS Tutorials, 2015.