2017 IEEE Computer Society Annual Symposium on VLSI

Hardware Acceleration for Machine Learning

Ruizhe Zhao, Wayne Luk, Xinyu Niu
*Department of Computing,
Imperial College London,
United Kingdom*

Huifeng Shi, Haitao Wang
*State Key Laboratory of Space-Ground
Integrated Information Technology,
Beijing,
China*

***Abstract*—This paper presents an approach to enhance the performance of machine learning applications based on hardware acceleration. This approach is based on parameterised architectures designed for *Convolutional Neural Network* (CNN) and *Support Vector Machine* (SVM), and the associated design flow common to both. This approach is illustrated by two case studies including object detection and satellite data analysis. The potential of the proposed approach is presented.**

1. Introduction

Machine Learning is a thriving area in the field of *Artificial Intelligence*, which involves algorithms that can learn and predict through, for example, building models from given datasets for training. In recent years, many machine learning techniques, such as Convolution Neural Network (CNN) and Support Vector Machine (SVM), have shown promise in many application domains, such as image classification and speech recognition.

Many hardware technologies can be used in accelerating machine learning algorithms, such as *Graphics Processing Unit* (GPU) and *Field-Programmable Gate Array* (FPGA). In particular, FPGA is a promising technology due to its low power consumption, reconfigurability, and real-time processing capability.

There are, however, two challenges for effective FPGA development. First, architectural descriptions should capture families of designs with different trade-offs in capability, performance and energy efficiency. Second,Second, a design flow for multiple architectures should facilitate re-use and comparison of optimisation methods and tools.

This paper presents a novel approach to address these challenges. The major contributions are as follows:

1) Parameterised hardware architectures for two well-known machine learning algorithms, *Convolution Neural Network* (CNN) and *Support Vactor Machine* (SVM).
2) A common design flow for different architectures designed for different algorithms and models.
3) Two case studies illustrating our approach, with a CNN design for object detection, and an SVM design for hyperspectral image classification.

Evaluations show that our approach is applicable to accelerating different machine learning algorithms and can reach competitive performance.

This paper has four main sections. Section 2 presents background in CNN and SVM, and also related research. Section 3 illustrates the basic architectures that are parameterised for machine learning tasks. Section 4 proposes a common design flow for given architectures. Section 5 shows two case studies of our approach.

2. Background

2.1. Convolutional Neural Networks

Many machine learning designs are based on deep learning networks, which involve an architecture with neurons grouped into layers by their functionalities, and multiple layers organised to form a deep sequential structure. Our focus is on *Convolutional Neural Network* (CNN), a classic deep learning network which has been applied to many vision-based tasks. A typical CNN contains the following layers:

- A convolution layer performs multi-dimensional convolution computation, which extracts features from an input feature map (fm_{in}) and generates a feature map (fm_{out}) with new features.
- An FC (Fully-Connected) layer usually performs the classification tasks at the end of a CNN. It applies *affine transformation* to the input feature map. An FC layer can be implemented with matrix-vector multiplication.
- Sub-sampling is a layer that can significantly reduce the dimensions of feature maps and enhance the *translation-invariance* property of CNNs. To acquire non-linearity, CNN usually contains activation layers, which are non-linear functions. Normalisation layers in CNNs often guarantee the probabilistic distribution will not change between the input and the output.

There are four well-known CNN architectures in recent years. AlexNet [1] is the first deep CNN architecture for large-scale image classification tasks. VGGNet [2] further increases the depth of CNN and achieves better performance than AlexNet. Inception [3] is a CNN that contains *inception* modules, which increases the computation efficiency of CNN based on irregular kernels. ResNet [4] has the best image classification performance by adopting residual shortcut connections in CNNs.

CNN-based Object Detection. CNN is an excellent feature extractor for vision-based tasks, such as object detection, which first targets *Regions of Interest* within an image and then assigns it with a class label. These two steps can be greatly enhanced by CNN, which has been discovered in Faster R-CNN [5] and YOLO [6]. We will discuss how CNN-based object detection algorithms can be accelerated by FPGA in Section 5.

2.2. SVM and Hyperspectral Image Classification

SVM Principles. Support Vector Machine (SVM) is a classic machine learning algorithm for classification and regression problems. The fundamental idea behind SVM is to find a hyperplane that can separate two groups of input data points, which should also be mapped to the same high-dimensional space as the hyperplane. The mapping from the original space to the higher-dimensional space is commonly implemented through a kernel function, which measures the distance between two vectors in the higher-dimensional space.

Multi-class SVM. Regarding multi-class classification, a frequently used method is constructing multiple binary SVM classifiers, which is known as *One-Versus-One* (OVO). In this approach, suppose the number of classes in the problem is K, then the number of binary SVMs should be $\frac{K(K-1)}{2}$, and their outputs will further construct a vector that can be measured by Hamming distances.

Hyperspectral Image Classification. Unlike a normal RGB image, a hyperspectral image (HSI), frequently used in satellite imaging tasks, covers information from across the electromagnetic spectrum in each pixel. Multi-class SVM is widely used for HSI classification because it can effectively deal with the Hughes phenomenon, caused by the high dimensionality of HSI data [7]. More details will be discussed in Section 5.

2.3. Related Work

CNN Accelerator Design. Design Space Exploration is a frequently used technique when optimising CNN hardware design. Zhang et al. [8] use the *roofline model* [9] and data dependence analysis to optimise a convolution-only AlexNet-based CNN architecture. Data *quantization* is a precision optimisation method that allows CNNs to use low-precision data types rather than floating-point. Qiu et al. [10] successfully deploy the VGG-16 architecture on an embedded FPGA platform with low-precision. Umuroglu et al. [11] explore binarised neural networks, which is an extreme case of quantization for FPGA platforms. Weight matrices in CNN models can be pruned and truncated until they become sparse. Han et al. [12] design a general *Efficient Inference Engine* (EIE) that computes based on compressed deep neural network models and takes into account the sparsity due to compression.

SVM Acceleration on FPGA Platforms. Irick et al. [13] optimise an SVM with a Gaussian Radial Base kernel function by using signed logarithmic number system. Papadonikolakis et al. [14] and Kyrkou et al. [15] present scalable FPGA architectures based on a cascaded SVM classifier scheme, which can effectively exploit FPGA resources and reconfigurability. Shao et al. [16] exploit the performance of incremental SVM training through an optimised dataflow architecture.

3. Parameterised Architectures

This section presents two parameterised architectures designed for CNN and SVM, with a focus on their common architectural parameters and scalable components. These two architectures are inspired by [17] and [18].

3.1. Overview

Hardware accelerators for different machine learning methods are, unsuprisingly, based on different architectures and building blocks. The following explores the parameters that can be used for such architectures and building blocks to achieve the desired trade-off in capability, performance, and energy efficiency.

There are two typical approaches to enhance the performance of an FPGA design: increasing the level of parallelism and revising the bit width of the data representation. To increase the level of parallelism, multiple processing units for the same functionality can be adopted. Regarding the bit width of the data representation, shorter bit width will lead to less resource usage of a building block and higher level of parallelism, but can also lead to unpredictable effects on the accuracy of the machine learning algorithm. Thus, there is usually a tradeoff between having higher accuracy or higher processing rate. This tradeoff can also be studied by simulating the performance with different data representations.

The two architectures presented in this section take the above considerations into account. They both contain parallelisable building blocks and flexible data types. Their descriptions are parameterised, such that the number of parallel building blocks and the bit width of the data representation are parameters. In the following discussion, we will focus on the parameters and their effects on our architectures. Section 4 will show a common design flow based on similar parameters and structures.

3.2. CNN Accelerator Architecture

The accelerator architecture for CNN is a streaming architecture: it takes input feature maps and weight matrices

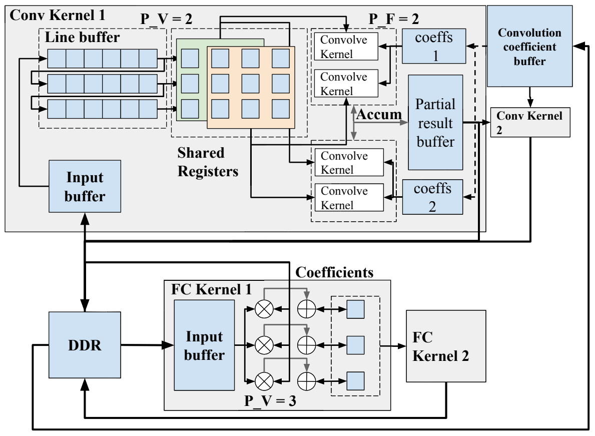

Figure 1. CNN accelerator architecture [17].

from off-chip memory and performs CNN computation by streaming data through each building block. There are two primary types of building blocks in this architecture: the `CONV` kernel that acts as a convolution layer, and the `FC` kernel that performs fully-connected computation. In either type of kernels, there are buffers to cache inputs and coefficients to improve data reuse, and computation units to perform specific types of computation, such as convolution and dot-product (Figure 1).

There are two levels of parallelism in this architecture. First, at the **computation unit** level, arithmetic circuits can be replicated to produce more results in each cycle. In the `CONV` kernel, it is possible to parallelise the computation either within a channel or among filters. For the first case, P_V^{conv} number of adjacent sliding windows for convolution can be computed in parallel, enabled by a *line buffer* and a group of *shared registers*. For the second case, P_F^{conv} number of output filters can also be computed in parallel, as their results are independent. In the `FC` kernel, P_V^{fc} number of dot-product operations can be run in parallel.

Second, at the **kernel** level, there can be multiple `CONV` (P_N^{conv}) and `FC` (P_N^{fc}) kernels placed on the same FPGA device if there is sufficient space. The data representation T does not need to be fixed in this architecture. Table 1 summarises the parameters used to describe a CNN architecture. Section 4 presents how these parameters affect the resource usage and performance.

3.3. SVM Accelerator Architecture

The architecture for accelerating SVM computation has several features similar to the CNN accelerator. First, it is a streaming architecture, data are loaded from input interfaces and on-chip preloaded ROMs and then processed by pipelined computation kernels, including the *Binary Classifier* (BC) kernels, Hamming distance computation kernels, and the Collection Kernel. Specifically, each BC kernel outputs classification information for one pixel and one of the category pairs; each Hamming distance kernel takes vectors from K BC kernels' output and computes the Hamming distance among $\frac{K(K-1)}{2}$ elements; while the Collection Kernel produces the final decision output.

Second, there are also building blocks that can be processed in parallel. This SVM accelerator architecture has two parameters related to the levels of parallelism, P_M^{SVM} specifies the number of image pixels to be processed in parallel, and P_N^{SVM} (equals to K) indicates the number of BC kernels to process one pixel. Thus the number of BC kernels is $P_M^{SVM} \times P_N^{SVM}$. The data type T of the SVM model coefficients is configurable to support the trade-off between model accuracy and design parallelism. Figure 2 illustrates this architecture while Table 2 summarises its parameters. Section 4 shows how to derive the resource usage and performance of this architecture.

TABLE 1. PARAMETERS FOR THE CNN ACCELERATOR ARCHITECTURE

Notation	Description
H^{conv}	Height of the convolution input feature map
W^{conv}	Width of the convolution input feature map
C^{conv}	Number of input channels
F^{conv}	Number of output filters
K^{conv}	Size of convolution kernels
P_V^{conv}	Number of parallel windows within one channel
P_F^{conv}	Number of parallel filters
P_N^{conv}	Number of parallel `CONV` kernels
P_V^{fc}	Number of parallel dot-product operations
P_N^{fc}	Number of parallel `FC` kernels
T	Data type

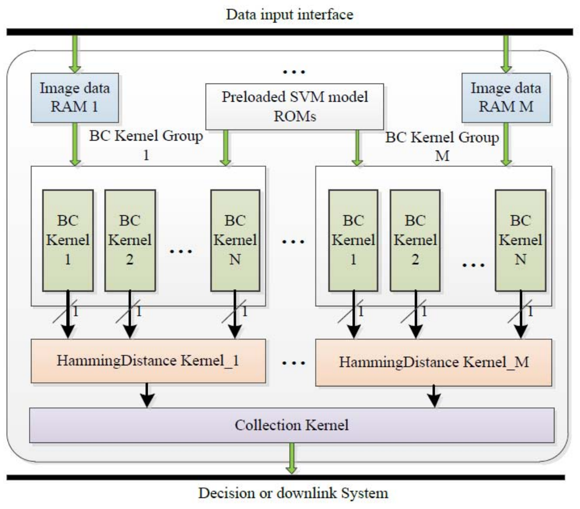

Figure 2. SVM accelerator architecture [18].

4. A Common Design Flow

This section presents a design flow common to architectures of both CNN and SVM. As mentioned in Section 3, these two architectures have similar structures and parame-

TABLE 2. PARAMETERS FOR THE SVM ACCELERATOR ARCHITECTURE

Notation	Description
K^{SVM}	Number of classes to be decided
l^{SVM}	Number of support vectors within each BC kernel
n^{SVM}	Number of dimensions in each support vector
P_M^{SVM}	Number of parallel pixel operations
P_N^{SVM}	Number of parallel BC kernels
T^{SVM}	Data type

ters, such as parallel units and data type. The flow illustrated in this section makes use of these common parameters: it makes use of a design model to predict the resource usage and performance based on these parameters. A constrained optimisation problem is formulated with the predicted performance as an objective function and resource usage as constraints, and the proposed optimisation module generates hardware designs for different applications.

4.1. Design Model

Resource Usage. There are three types of resources in an FPGA device: logic that includes *Look Up Tables* (LUTs) and *Flip-Flops* (FFs), *Block RAMs* (BRAMs) that act as on-chip memory, and *Digital Signal Processors* (DSPs) that contain high-performance arithmetic units. The following focuses on the relationships among resource usage and parameters that are tunable. Parameters that originate from application specifications are constants and will not be considered in this section. There are many ways to resolve these constants, such as linear regression on the resource usage of generated designs. We will use the big-O notation to reflect the key relationships. Table 3 shows the resource usage models for our architectures. Several discoveries are as follows:

- Although under big-O notations logic usage and DSP usage are the same, they cannot share the same set of constants.
- The BRAM usage in CNN is mainly due to the partial result buffer, which is linear to the number of filters computing in parallel. The BRAM usage in SVM is from image pixels RAM and preloaded SVM model ROM, which is only affected by the number of parallel pixel operations.
- The impacts of different data types are reflected in the constants of the design models.

Performance. The performance of a generated design has two aspects: *speed* ($\mathcal{S}$) and *power consumption* ($\mathcal{P}$). This paper adopts *throughput* to indicate the speed of the design. We assume that the speed is linear to the parallel processing units. For power consumption, the power consumption of an FPGA design can be divided into *static power* and *dynamic power* components, which are proportional to the on-chip space usage and the clock frequency respectively. Given the clock frequencies for the CNN and SVM designs are $\mathcal{F}^{CNN}$

TABLE 3. RESOURCE USAGE FOR ARCHITECTURES TO ACCELERATE CNN AND SVM

Type	Resource Usage Model
$\mathcal{U}_{logic}^{CNN}$	$O(P_N^{conv}P_V^{conv}P_F^{conv}) + O(P_N^{fc}P_V^{fc})$
$\mathcal{U}_{bram}^{CNN}$	$O(P_N^{conv}P_F^{conv})$
$\mathcal{U}_{dsp}^{CNN}$	$O(P_N^{conv}P_V^{conv}P_F^{conv}) + O(P_N^{fc}P_V^{fc})$
$\mathcal{U}_{logic}^{SVM}$	$O(P_N^{SVM}P_M^{SVM})$
$\mathcal{U}_{bram}^{SVM}$	$O(P_M^{SVM})$
$\mathcal{U}_{dsp}^{SVM}$	$O(P_N^{SVM}P_M^{SVM})$

TABLE 4. PERFORMANCE ESTIMATION FOR ARCHITECTURES TO ACCELERATE CNN AND SVM

Type	Performance Model
$\mathcal{S}^{CNN}$	$O(\mathcal{F}^{CNN}(P_N^{conv}P_V^{conv}P_F^{conv} + P_N^{fc}P_V^{fc}))$
$\mathcal{P}^{CNN}$	$O(\mathcal{F}^{CNN}) + O(P_N^{conv}P_V^{conv}P_F^{conv} + P_N^{fc}P_V^{fc})$
$\mathcal{S}^{SVM}$	$O(\mathcal{F}^{SVM}P_N^{SVM}P_M^{SVM})$
$\mathcal{P}^{SVM}$	$O(\mathcal{F}^{SVM}) + O(P_N^{SVM}P_M^{SVM})$

and $\mathcal{F}^{SVM}$, we could derive the performance models in Table 4.

4.2. Optimisation Problem Formulation

To get the values for architectural parameters with the best performance, we present a constrained optimisation problem, in which the objective function is based on performance models while the constraints involve resource usage models. Intuitively, the objective function for a hardware design is a combination of its speed and power consumption: the speed should be as high as required, and the power consumption should be as low as possible. Given the parameter alg can either be CNN or SVM, we assume the objective function has the form $\alpha(\mathcal{S}^{alg}) - \beta(\mathcal{P}^{alg})$, in which α and β are two positive hyper-parameters that specify weights for these two performance metrics. Constraints for this objective function capture the condition that the predicted resource usage should be less than the resource capacity on the FPGA device. The final form of the optimisation problem is shown in Equation 1. Note that all the constants should be resolved to specific values in order to get a reasonable answer from the solution of the problem.

$$\begin{array}{lrcl} \max & (\alpha\mathcal{S}^{alg} & - & \beta\mathcal{P}^{alg}) \\ \text{subject to} & \mathcal{U}_{logic}^{alg} & \leq & \mathcal{U}_{logic}^{max} \\ & \mathcal{U}_{bram}^{alg} & \leq & \mathcal{U}_{bram}^{max} \\ & \mathcal{U}_{dsp}^{alg} & \leq & \mathcal{U}_{dsp}^{max} \end{array} \quad (1)$$

Various contraint solvers should be able to produce solutions to the above constraint optimisation problem. Recently it is proposed that machine learning techniques, such as transfer learning, can also be used in optimising parameters for many applications targeting FPGA-based implementations [19].

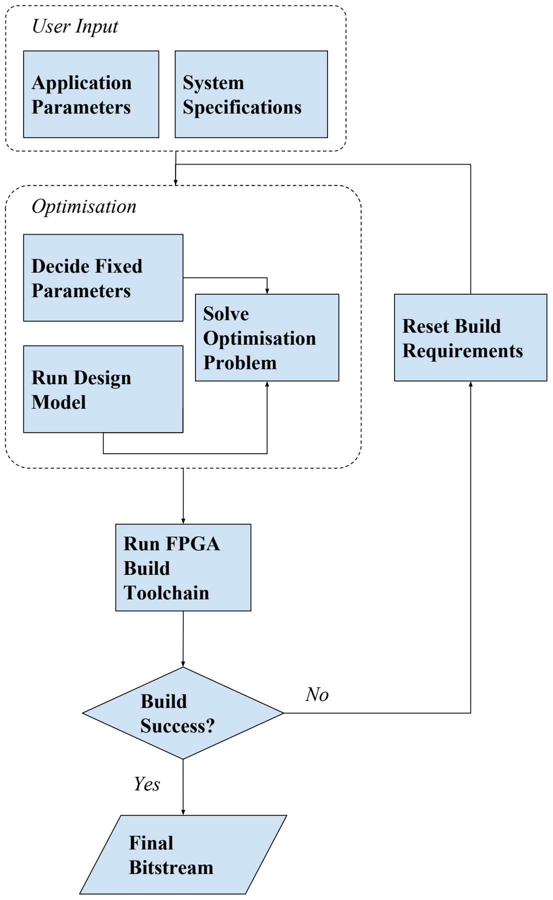

Figure 3. A common design flow for different machine learning architectures.

4.3. Design Flow

Figure 3 shows the proposed design flow. At the beginning of the flow, users need to specify the system requirements, such as the clock frequency and the percentage of hardware resource they would like to use. Parameters for applications should also be specified. Next in the flow is the optimisation module that is discussed in Section 4.2, which takes user inputs to solve the optimisation problem. Note that only the parameters relevant to the optimisation problem are important in this module; other application related parameters will be fixed, such as the height and width of the convolution layer. Also, although the output of the optimisation module meets resource constraints of the system, there are several other constraints that can only be addressed after running the place and route algorithm, such as the *critical path delay* constraint. So there is a backwards flow after the building process unless the build succeeds. As this flow has no direct dependency on specific algorithm or architecture, it can be applied to many other FPGA-based machine learning applications.

TABLE 5. PERFORMANCE EVALUATION FOR CNN-BASED OBJECT DETECTION

	ARM CPU	FPGA	GPU
Platform	ARMv7-A	ZC706	Titan X
Technology	28 nm	24 nm	16 nm
Clock Freq.	Up to 1 GHz	200 MHz	1531 MHz
Num. of Cores	2 cores	—	—
YOLO	430.6 s	0.744 s	0.010 s
Faster R-CNN	Failed	0.875 s	0.062 s
YOLO	1.6 W	1.167 W	230 W
Faster R-CNN	Failed	1.167 W	81 W
YOLO	688.96 J	0.868 J	2.30 J
Faster R-CNN	Failed	1.02 J	5.02 J

5. Case Studies

This section evaluates the effectiveness of our proposed approach based on two applications: CNN-based object detection and SVM-based hyperspectral image classification. Our results show that the approach presented in this paper can achieve competitive performance.

5.1. CNN-based Object Detection

As mentioned in Section 2, there are two well-known CNN-based object detection algorithms: YOLO and Faster R-CNN. The CNN architectures of YOLO and Faster R-CNN that we evaluate are Inception and VGG-16 respectively. The target platform of our evaluation is the Xilinx Zynq (ZC706) embedded FPGA platform, which contains moderate resources on its FPGA board. After setting the resource capacity and resolving the constants in our models, we derive an optimised configuration of the tunable parameters. In this design, we deploy 1 `CONV` kernel with $P_V^{conv} = 4$ and 1 `FC` kernel with $P_V^{fc} = 4$. The data type used is 32-bit fixed-point, which is chosen for detection accuracy. Table 5 shows the final evaluated results. Compared with the ARM CPU version, the generated FPGA design is much faster. When compared with the GPU version (2.3J), the energy consumption of the FPGA design (0.868J) is much smaller.

5.2. SVM-based Hyperspectral Image Classification

We also evaluate the SVM architecture for hyperspectral image classification, often used in satellite image processing. Our accelerator is implemented on a Maxeler MAX4 DFE which is equipped with a Stratix V FPGA. We use the AVIRIS HSI data set to train and evaluate the performance of SVM classifiers. By applying our design flow, we find that $P_N^{SVM} = 15$, $P_M^{SVM} = 8$, with 16-bit fixed-point data type in the final design. Our results show that this design can use more than 80% logic and DSPs and 66.8% BRAM. Also, we compare its performance with other processors, such as

TABLE 6. PERFORMANCE EVALUATION FOR SVM-BASED HSI CLASSIFICATION

	Intel CPU	ARM CPU	DFE
Platform	E5–2650	Cortex A9	Max4
Technology	32 nm	28 nm	28 nm
Clock Freq.	2.00 GHz	Up to 1 GHz	120 MHz
Num. of Cores	8 cores	2 cores	-
Time (μs/pixel)	25.8	1321.2	0.99
Power (W)	95	3.3	26.3
Energy (J/pixel)	2.45×10^{-3}	4.36×10^{-3}	2.60×10^{-5}
Relative run time	14.2	1334.5	1.0

an Intel Xeon E5-2650 CPU and an ARM Cortex A9 CPU. Table 6 shows that our design has competitive performance. Compared with the results on Xeon and Cortex, our DFE design has at least 14.2 times speed up and lower power consumption.

5.3. Discussion

These two case studies have shown that our architectures and design flow and support high performance. The architectures are parameterised, FPGA platforms with different resource capacities, such as Xilinx ZC706 and Maxeler Max4 DFE. The performance of these designs is competitive compared with software running on CPU and GPU, in terms of speed and/or power consumption. These two case studies show that other applications can be accelerated on FPGA as long as they adopt similar architectural parameters.

6. Conclusion

This paper describes how hardware acceleration can enhance the performance of machine learning applications. It is based on parameterised architectures for such applications, and the associated optimisation techniques and flow. The proposed approach is illustrated by two examples: CNN-based object detection, and SVM-based hyperspectral image classification. Future work includes extending our approach to cover further applications based on CNN and SVM, and other machine learning algorithms such as Gaussian Mixture Model [20] and Genetic Algorithm [21]. Further FPGA-based optimisations such as *multi-pumping* [22] will also be included in the proposed design flow.

Acknowledgments

The support of the European Union Horizon 2020 Research and Innovation Programme under grant agreement number 671653, UK EPSRC (EP/L00058X/1, EP/L016796/1, EP/N031768/1 and EP/P010040/1), the State Key Laboratory of Space-Ground Integrated Information Technology, Intel, Maxeler and Xilinx, Inc. is gratefully acknowledged.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*. IEEE Computer Society, 2016, pp. 770–778.

[5] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[7] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Transactions on geoscience and remote sensing*, vol. 42, no. 8, pp. 1778–1790, 2004.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

[9] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[10] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," in *FPGA*, 2016.

[11] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2017.

[12] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *ISCA*. IEEE Computer Society, 2016, pp. 243–254.

[13] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A hardware efficient support vector machine architecture for FPGA," in *FCCM*, 2008.

[14] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade fpga accelerator for support vector machines classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2012.

[15] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*. IEEE, 2013.

[16] S. Shao, O. Mencer, and W. Luk, "Dataflow design for optimal incremental svm training," in *FPT*, 2016.

[17] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing cnn-based object detection algorithms on embedded fpga platforms," in *ARC*, 2017.

[18] S. Wang, X. Niu, N. Ma, W. Luk, P. Leong, and Y. Peng, "A scalable dataflow accelerator for real time onboard hyperspectral image classification," in *ARC*, 2016.

[19] M. Kurek, M. P. Deisenroth, W. Luk, and T. Todman, "Knowledge transfer in automatic optimisation of reconfigurable designs," in *FCCM*, 2016.

[20] C. Guo, H. Fu, and W. Luk, "A fully-pipelined expectation-maximization engine for Gaussian mixture models," in *FPT*, 2012.

[21] L. Guo, C. Guo, D. B. Thomas, and W. Luk, "Pipelined genetic propagation," in *FCCM*, 2015.

[22] R. Zhao, T. Todman, W. Luk, and X. Niu, "DeepPump: Multi-pumping deep neural networks," in *ASAP*, 2017.