Journal of Systems Architecture 97 (2019) 278–286

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Designing efficient accelerator of depthwise separable convolutional neural network on FPGA

Wei Ding[a,b], Zeyu Huang[a,b], Zunkai Huang[a], Li Tian[a], Hui Wang[a,*], Songlin Feng[a]

[a] Shanghai Advanced Research Institute, Chinese Academy of Sciences, Shanghai 201210, P.R. China
[b] University of Chinese Academy of Sciences, Beijing 100049, P.R. China

ARTICLE INFO

Keywords:
Convolutional neural network
Depthwise separable convolution
Hardware accelerator
FPGA
Edge computing

ABSTRACT

In recent years, convolutional neural networks (CNNs) have achieved state-of-the-art results for many computer vision tasks. However, the traditional CNNs are computational-intensive and memory-intensive, hence they are unsuitable for the application in mobile edge computing scenarios with limited computing resources and low power consumption. The depthwise separable CNNs can significantly reduce the number of model parameters and improve the calculation speed, so it is naturally suitable for mobile edge computing applications. In this paper, we propose a Field Programmable Gate Array (FPGA)-based depthwise separable CNN accelerator with all the layers working concurrently in a pipelined fashion to improve the system throughput and performance. To implement the accelerator, we present a custom computing engine architecture to handle the dataflow between adjacent layers by using double-buffering-based memory channels. Besides, in fully connected layers, data titling technique is adopted to divide matrix multiplication from large dimension into small matrix. Finally, our proposed accelerator for depthwise separable CNN has been implemented and evaluated on Intel Arria 10 FPGA. The results of experiment indicate that the proposed depthwise separable CNN accelerator has a performance of 98.9 GOP/s and achieve up to 17.6× speed up and 29.4× low power than CPU and GPU implementations respectively.

1. Introduction

Convolutional neural networks (CNNs) have been widely used in computer vision applications, such as image classification [1,2], object detection [3,4], image segmentation [5,6]. The high accuracy and good performance of CNNs have promoted the development of medical image analysis [7], face recognition [8] and autonomous driving technology [9,10].

In general, the state-of-the-art CNN models run within a web service in the cloud, and this traditional operation mode often encounters problems like enormous computational complexity and insufficient memory resources. There is an urgent demand for embedding the deep neural networks into the mobile edge computing scenarios, such as robotics, drones, and autonomous driving, where there are strict limitations regarding physical size, energy consumption and low latency. Therefore, developing compact neural networks with small model size and high efficiency is required desperately.

Overall, there are two practical approaches. one approach is called model compression [11–13], including data quantization [11], singular value decomposition (SVD) [12], pruning and Huffman coding [13]. This approach can reduce the number of operation and memory footprint for mobile devices by compressing the pre-trained networks, but the compression ratio is always limited. Another approach is to directly design and train the compact networks, such as SqueezeNet [14], Xception [15] and MobileNet [16,17]. Xception and MobileNet utilizes the depthwise separable convolution [18] to build a compact and efficient neural network models for mobile vision application. The depthwise separable CNNs usually hold advantages of lower computational complexity and less parameters while maintaining impressive performance. For example, compared to the VGG-16 [2], the MobileNet v1 [16] based on depthwise separable convolution achieves up to 32× less parameters and 27× less operations, while the accuracy only decreases 0.9%. Consequentially, the depthwise separable CNNs with low latency and small model size could meet the demands of mobile and embedded vision applications. Additionally, the approach of designing the compact networks e.g. depthwise separable CNNs is orthogonal to model compression. Therefore, by further applying the model compression approach to the pre-trained depthwise separable CNNs, more compact and efficient networks could be developed.

Due to the intrinsically parallel computing characteristics of CNNs, CPU is not efficient enough to perform CNN calculation, so it is difficult to meet the performance requirements in the algorithm training and inference stages. To address this problem, various accelerators have been proposed, such as Graphics Processing Unit (GPU) [19], FPGA [20], and

* Corresponding author.
E-mail addresses: dingwei@sari.ac.cn (W. Ding), wanghui@sari.ac.cn (H. Wang).

https://doi.org/10.1016/j.sysarc.2018.12.008
Received 2 October 2018; Received in revised form 29 November 2018; Accepted 27 December 2018
Available online 28 December 2018

ASIC [21,22]. GPU is currently the most used accelerator for deep learning training in cloud computing. Unfortunately, GPU's natural of high power consumption inevitably limits its application in edge computing scenarios, where low power consumption and low latency are required strictly.

FPGA can provide thousands of programmable logic blocks and configurable interconnections, which enables the construction of custom-tailored accelerator architectures in hardware for high performance computing [23] and machine vision [24,25]. Recently, CNN acceleration scheme based on FPGA has attracted a growing number of researchers [20,26,27]. Compared with the CPU-based and GPU-based CNN acceleration scheme, the FPGA-based scheme shows its unique advantages like high performance, high energy efficiency and low latency. However, researchers now mainly focus on the acceleration of standard CNNs and they pay little attention on the FPGA-based acceleration for compact networks, especially for the depthwise separable CNNs. In this work, we propose an efficient accelerator for depthwise separable convolutional neural network on FPGA for mobile edge computing.

The key contributions of this work are summarized as follows:

- we propose a depthwise separable convolutional neural network accelerator with all the layers working concurrently in pipelined fashion to improve the system throughput and performance.
- We propose custom computing engine architectures for depthwise convolution and pointwise convolution, which compose the depthwise separable convolution computing engine.
- As a case study, we implement a depthwise separable CNN accelerator on Arria 10 FPGA with the proposed architecture and optimizing strategies. The depthwise separable CNN accelerator can achieve a peak performance with 98.91 GOP/s and 15.57 μs execution time per image with the size of $32\times32\times3$ under 180 MHz maximum clock frequency in a pipelined style, and it shows promising improvements over CPU and GPU baselines.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 briefly describes the preliminaries of depthwise separable CNNs. Section 4 presents the architecture of depthwise separable CNN accelerator. Some optimizing strategies are given in Section 5. The performance of the proposed accelerator of depthwise separable CNN is evaluated in Section 6. We finally conclude this paper in Section 7.

2. Related work

To accelerate the CNN execution for mobile edge computing, various specific hardware accelerators on FPGA are proposed to enhance the data reuse, memory access efficiency and convolution operations [20,28,29]. Some FPGA-based CNN accelerators only focus on optimizing the convolution computing engines without considering the memory transfers [28,29]. References [28,30] exploits different parallelism strategies to improve the performance, such as the parallelism within feature maps, convolution kernel and loop unrolling. Some other works not only focus on optimizing the computational resource but also consider the optimizing of memory transfers [20,31,32]. Those works proposed a design space exploration framework that can evaluate and explore various architecture choices by optimizing both computation resources and memory access for FPGA-based CNN accelerators. Besides, references [27,33] propose inter-layer pipelined architecture for CNN accelerator. The work [27] proposes an end-to-end FFPGA-based CNN accelerator with all the layers mapped on one chip. The work [33] adopts a deeply pipelined architecture using FPGA cluster instead of single-board FPGA, which expands the design space for optimal performance and energy efficiency. Nevertheless, most of above-mentioned works are only suitable for standard convolutional neural networks such as AlexNet [1] and VGG-16 rather than the depthwise separable convolution neural networks. The work [34] presented an FPGA acceleration system for a redundancy-reduced MobileNet which is based on depthwise separable convolution. This work proposes a pipelined architecture with all the layers mapped on one chip and this architecture is optimized for depthwise separable convolutional neural networks accelerator.

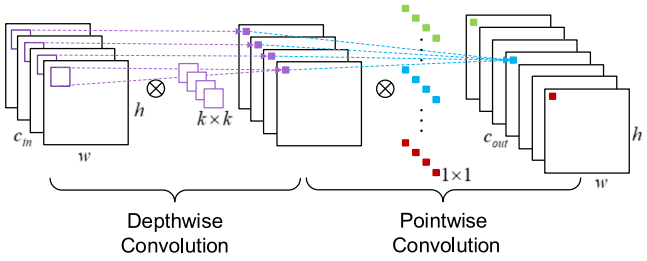

Fig. 1. Depthwise separable convolution. Including depthwise convolution and pointwise convolution.

3. Preliminaries of depthwise separable convolution neural networks

3.1. Depthwise separable convolution primer

The depthwise separable convolution was originally introduced by Laurent Sifre [18] and was applied in the study of image classification. The depthwise separable convolution is a form of factorized convolution which factorizes a standard convolution into a depthwise convolution and pointwise convolution. As shown in Fig. 1, the depthwise convolution performs lightweight filtering by applying a single filter per input channel. The pointwise convolution then apples a 1×1 convolution to combine linearly the input channels. The depthwise separable convolution replaces the standard convolution operation with a factorized two-layer convolution, one layer for space filter and one layer for combining. Hence depthwise separable convolution can drastically reduce the redundant computation and mode size.

A standard convolutional layer takes an $h\times w\times c_{in}$ input feature map **I**, and applies a $k\times k\times c_{in}\times c_{out}$ convolutional kernel **K** to produce an $h\times w\times c_{out}$ output feature map **O** where h and w are the height and width of input feature maps, k is the spatial dimension of the kernels assumed to be square, c_{in} is the number of input channels and c_{out} is the number of output channels. In this paper, it is assumed that the output feature maps can get the same spatial size as the input feature maps by zero-padding operation. Standard convolution has a computational complexity (the operation numbers of Multiplication and Accumulation, MAC) of $k^2\cdot c_{in}\cdot c_{out}\cdot h\cdot w$.

The depthwise separable convolution consists of two parts: depthwise convolution and pointwise convolution. We use depthwise convolution to apply a single filter per each channel of input feature maps. The depthwise convolution can be written as Eq. (1) [16].

$$\mathbf{G}(y,x,j)=\sum_{u=1}^{k}\sum_{v=1}^{k}\mathbf{K}(u,v,j)\times\mathbf{I}(y+u-1,x+v-1,j) \tag{1}$$

where **K** is the depthwise convolutional kernels of size $k\times k\times c_{in}$. The n_{th} filter in **K** is applied to the n_{th} channel in **I** to produce the n_{th} channel of the filtered output feature map **G**.

The pointwise convolution computes the linear combination of the output of depthwise convolution via 1×1 convolution to build new features. The pointwise convolution can be written as Eq. (2) [16].

$$\mathbf{O}(y,x,l)=\sum_{j=1}^{c_{in}}\mathbf{G}(y,x,j)\times\mathbf{P}(j,l) \tag{2}$$

where the size of 1×1 convolution kernel is $1\times1\times c_{in}\times c_{out}$. By adjusting m, we can change the number of channels in the output feature map. Compared with $k\times k(k>1)$ convolutional operations, the dense 1×1 convolutional operation has no the limitation of near locality, so

Table 1
Depthwise separable CNN (DS-CNN) architecture.

Layer	Type	Filter Shape (pixel)	Input Size (pixel)
Conv1	Conv	3×3×3×16	32×32×3
	Max pool	Pool 2×2	32×32×16
Conv2	DW_Conv	3×3×16	16×16×16
	PW_Conv	1×1×16×32	16×16×16
	Max pool	Pool 2×2	16×16×32
Conv3	DW_Conv	3×3×32	8×8×32
	PW_Conv	1×1×32×64	8×8×32
	Max pool	Pool 2×2	8×8×64
FC4	FC	1024×10	4×4×64
Classifier	Softmax	–	1×1×10

Table 2
Standard CNN (STD-CNN) architecture.

Layer	Type	Filter Shape (pixel)	Input Size (pixel)
Conv1	Conv	3×3×3×16	32×32×3
	Max_pool	Pool 2×2	32×32×16
Conv2	Conv	3×3×32	16×16×16
	Max_pool	Pool 2×2	16×16×32
Conv3	Conv	3×3×64	8×8×32
	Max_pool	Pool 2×2	8×8×64
FC4	FC	1024×10	4×4×64
Classifier	Softmax	–	1×1×10

it doesn't require to reorder the parameter in memory. Therefore, it can be implemented directly with highly optimized general matrix multiply algorithms.

The depthwise separable convolution computational complexity can be written as Eq. (3):

$$C_{ds} = k^2 \cdot c_{in} \cdot h \cdot w + c_{in} \cdot c_{out} \cdot h \cdot w \tag{3}$$

It is the sum of computation cost of the depthwise convolution and 1×1 pointwise convolution.

Compared with the standard convolution, depthwise separable convolution reduces the computational complexity by a factor as η, which can be expressed as the following Eq. (4).

$$\eta = \frac{C_{ds}}{C_{std}} = \frac{k^2 c_{in} hw + c_{in} c_{out} hw}{k^2 c_{in} c_{out} hw} = \frac{1}{c_{in}} + \frac{1}{k^2} \tag{4}$$

In general, the value of m is relatively large, so the factor η is approximately equal to $1/k^2$. This paper uses 3×3 depthwise separable convolution, so the computation complexity and the number of parameters of corresponding convolution layers are 7~8 times less than that of standard convolution at only a small accuracy loss as see in Section 3.2.

3.2. Depthwise separable convolution neural network instance

To compare the difference between the standard CNN and the depthwise separable CNN for hardware implementation, this paper designs two lightweight CNN architectures which are based on standard convolution and depthwise separable convolution, respectively.

Table 1 illustrates a CNN model based on a depthwise separable convolution (DS-CNN), starting with the standard convolutional layer (Conv) and the max pooling layer, followed by the depthwise convolution (DW_Conv), pointwise convolution (PW_Conv) and the max pooling (Max_pool) layer. After two consecutive depthwise separable layers and max pooling layers, there are fully connected (FC) layer and the softmax classifier. The convolution neural network based on standard convolution (STD-CNN) has the same structure and number of feature maps as DS-CNN, as shown in Table 2. All the above convolutional layers are followed by a ReLU nonlinear activation layer. The stride of convolution layers is 1 px and the max pooling layers is 2 px.

The computational complexity and the number of weights for each layer in the STD-CNN and DS-CNN model are collected as shown in Figs. 2 and 3. These two figures could reflect the computation-intensive feature of convolution layers and the memory-intensive feature of FC layers. Besides, it can see that the computational complexity and the number of weights in the convolutional layers based on depthwise separable convolution are reduced by 7.8 times and 7.4 times respectively than that of network based on standard convolution. Duo to the network architecture defined in this paper is relatively lightweight and only has 4 layers, the number of weights of the fully connected layer account for 75.1% of the whole network. when we design a deeper and wider network based on depthwise separable convolution, the computational cost and the number of weights will be reduced obviously.

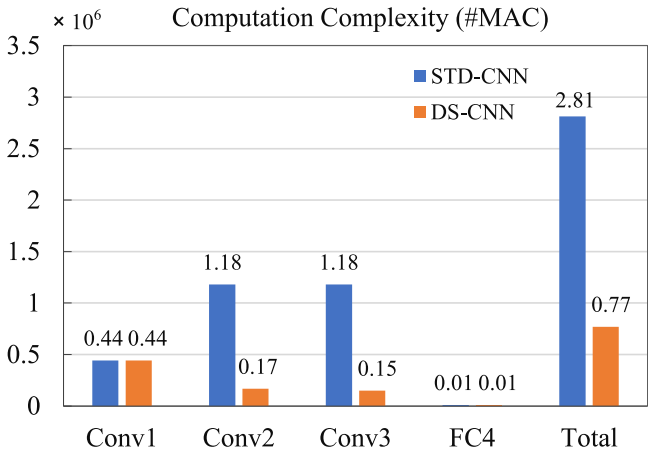

Fig. 2. Computational complexity for STD-CNN and DS-CNN.

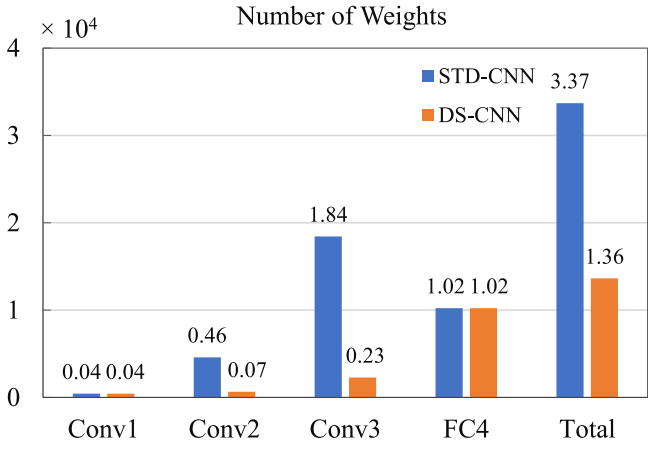

Fig. 3. Number of weights for STD-CNN and DS-CNN.

The STD-CNN and DS-CNN defined in this paper have been trained and tested in the MNIST dataset [35]. The accuracy of STD-CNN is 99.5%, and the accuracy of DS-CNN is 99.1%. Furthermore, on the CIFAR-10 dataset [36], the accuracy of STD-CNN is 76.4%, and the DS-CNN is 72.6%. Although there is a slight decrease in accuracy, the parameters of the convolutional layers are reduced by 7.4 times, and the overall computational cost is reduced by 3.7 times. If the parameters of neural network based on depthwise separable convolution are extended to be the same as STD-CNN, the accuracy on the CIFAR-10 dataset can reach 84.4%.

4. Accelerator design

4.1. Accelerator architecture overview

In the inference phase of CNN, one layer is calculated and then is used as the input of the next layer. In view of this continuous calcula-

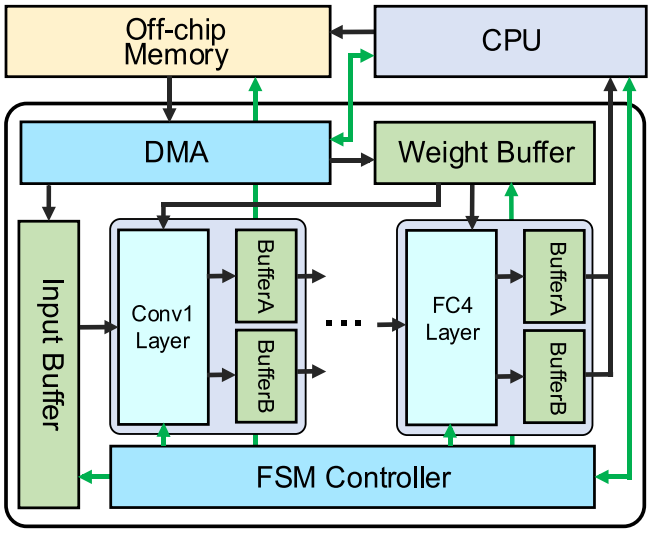

Fig. 4. An overview of the accelerator architecture.

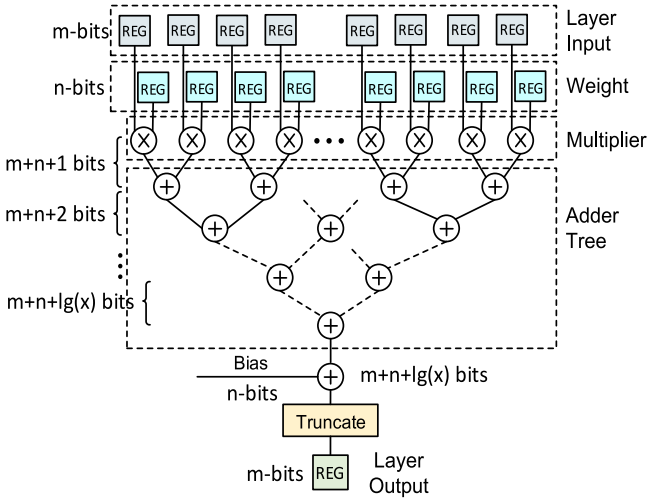

Fig. 6. MAC unit. In this work, the fixed-point word length m and n are 16 bits.

tion process, this paper proposes a pipeline-based depthwise separable CNN acceleratorarchitecture as shown in Fig. 4, taken DS-CNN as an example. In the proposed architecture, 3 Conv layers and 1 FC layer run simultaneously in a pipelined fashion. Besides, the intermediate results between layers are stored in the on-chip memory and the data access time can be reduced significantly, increasing the system throughput and performance.

Our system architecture consists of convolutional computing engines, on-chip buffers, a Finite State Machine (FSM) controller, a direct memory access (DMA) for off-chip memory data transfer, off-chip memory and CPU. The black lines are the data paths and the green lines are the control paths. The convolutional computing engines take charge of most of computation tasks of the convolutional layers, activation layers, pooling layers and fully connected layers in the CNN accelerator. The FSM controller mainly coordinates the pace of each convolutional layer, as well as the communication with the external CPU. The on-chip buffers consist of the input data buffers, weight buffers, intermediate data buffers and output data buffers. Benefiting from the development of CNN model compression methods and lightweight neural networks, the number of parameters and the computational complexity have been significantly reduced. In addition, the logical resources and on-chip memory on FPGA have improved a lot. For instance, the on-chip memory of Intel FPGA Arria 10 device reaches 53 Mb. The two reasons mentioned above make it possible to store the intermediate results between the layers in the on-chip memory and increase the system bandwidth.

4.2. Convolutional layer computing engine design

The convolutional layers are the computation-intensive part of the entire acceleration system. The final acceleration effect of the CNN mainly depends on the acceleration and optimization of convolutional layers. Fig. 5 is the schematic diagram of the depthwise separable convolutional layer computing engine. The computing engine mainly includes the input data buffer banks, the weight buffer banks, the depthwise convolution units (DCUs), the pointwise convolutional units (PCUs) and the output buffer banks.

The data path of convolutional layers and fully connected layer consist of a series of MAC operations (multiplication and accumulation, MAC), as shown in Fig. 6. The input data of input feature maps is multiplied with the weights of neural networks and the results are accumulation to form the layer output of output feature maps. For 2D convolution, the multiplication is performed by using parallel multipliers, and the adder tree sums all the results from multipliers in parallel per cycle with pipeline fashion. In Fig. 6, *m* and *n* refer to the number of bits for

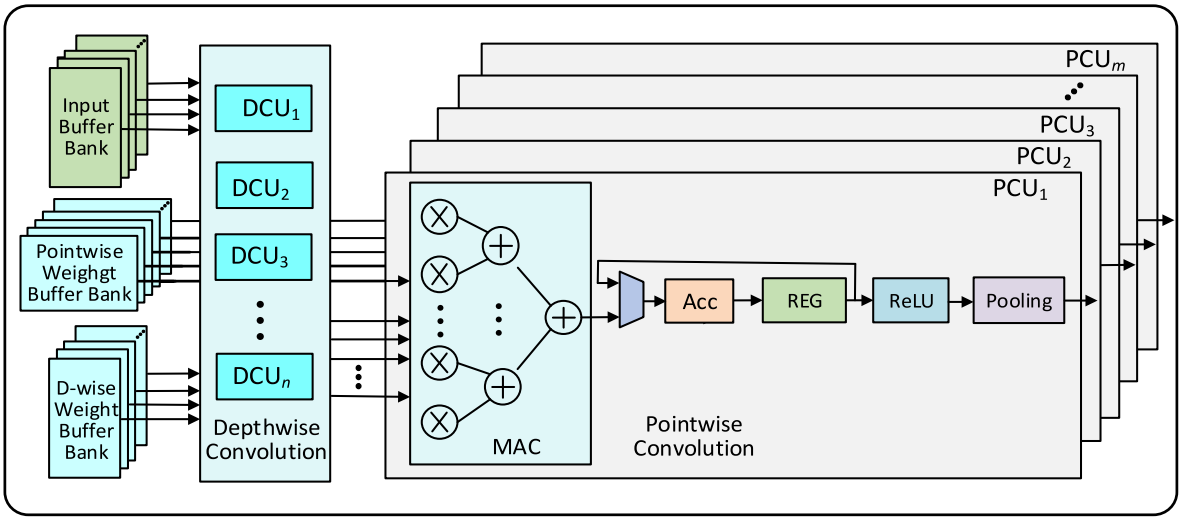

Fig. 5. Depthwise separable convolutional layer computing engine.

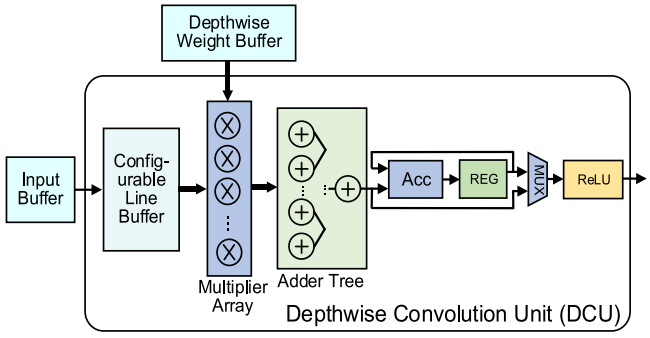

Fig. 7. Depthwise convolution unit.

layer inputs and weights, respectively. x is the number of elements in each layer input. Notice that we reduce the word width of layer output by truncating, so that the word width of layer outputs is the same as the layer inputs.

The depthwise convolution unit (DCU) is composed of the configurable line buffer and the MAC unit as shown in Fig. 7. The depthwise convolution is carried out by $k \times k$ convolution operations, which has the spatial proximity. To make fully use of the spatial proximity to reduce the access of input features, this paper employed a configurable line buffer. When input data goes through the buffer in row-major layout, the line buffer releases a sliding window selection on the input image and multiple rows of input pixels can be buffered for simultaneous access. Each DCU is responsible for the 2D convolutional computation of a single input feature map with the corresponding weights and multiple DCUs in parallel constitute the depthwise convolution. The DCU can also be configured to perform the standard convolution by using the Acc and MUX for partial summation. Different convolutional layers have different input channels and parameters, so we can design different numbers of DCUs with a parallel factor p_n according to different convolutional layers.

The pointwise convolution unit (PCU) consists of MAC computing unit, accumulators, no-linearity activation modules and max pooling modules as shown Fig. 5. MAC computing units perform 1×1 convolution by multiplying the output of depthwise convolution and the pointwise weights from the pointwise weight buffer bank, summing all the results from multipliers. Then, multi-channels results are accumulated by the previous accumulation value. The ReLU function is used for nonlinear activation, and the max pooling is applied to down-sampling. The pooling result is stored in the output buffer. The result of depthwise convolution is copied to multiple PCUs through broadcast operation and the parallel computation of the output feature maps is carried out by multiple PCUs with a parallel factor p_m simultaneously.

4.3. Fully connected layer design

There is no weight share and data reuse of the input feature maps in the fully connected layers (FC layers), thus a memory access is needed for every operation. Therefore, the design and optimizing target is to minimize the data access between off-chip memory and on-chip memory. The computation of FC layer is essentially a matrix multiplication implemented with M × V. The FC layer of DS-CNN in this paper has a 1024-dimensional input feature vector and has 10-dimensinal output feature vector, corresponding to 1024×10 of weight matrix. If the 1024-dimensional matrix multiplication is carried out in parallel, too many multipliers are needed to be consumed. On the contrary, if the 1024-dimensional matrix is performed in a serial way, the running time is too long to meet the system requirements. To address this problem, we employ a trade-off solution proposed in [27]. The matrix multiplication with a large dimension is divided into multiple small-scale matrix multiplications as shown in Fig. 8.

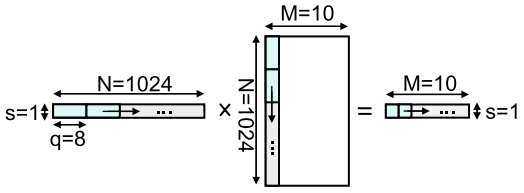

Fig. 8. Weight window in the weight matrix shifts vertically.

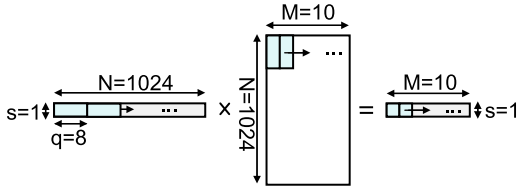

Fig. 9. Weight window in the weight matrix shifts horizontally.

As shown in Fig. 8, the 1024-dimensional input feature vector is divided into $p = 128$ small scale vectors, every vector has $q = 8$ dimensions. In the weight matrix, the decomposition way of each column is the same as that of the input feature vector, and the priority is given to move vertically. The small matrix can be computed in parallel, so that only 128 cycles are required to calculate a column to get a result. After one column is completed, the next column is carried out. However, the above method does not consider data reuse, and every small block matrix calculation requires repeated data access of the input feature matrix. To reduce the number of data access, this paper proposes a method of horizontal row priority computing pattern as shown in Fig. 9. The input tile matrix and the $M = 10$ tile matrices in the weight matrix are performed simultaneously every cycle, and each tile is calculated in parallel, which can speed up the matrix calculation and reduce the data read times. Only 128 cycles are needed to complete the calculation of the entire matrix. The basic computing unit of matrix multiplication adopts the MAC computing unit as shown in Fig. 6.

4.4. Memory system design

To accommodate multiple reads and writes per cycle and feed the DCUs and PCUs with data efficiently, we design the data buffer structure on the Convolutional Computing Engine side, the Input Buffer and the Output Buffer, as shown in Fig. 10. Our solution uses 2 ping-pong buffer bank A and buffer bank B with equal size for each stage, where the former layer may write to or read from buffer bank A while the next layer reads from buffer bank B. Each Input Buffer Bank has a lot of parallel RAMs and the number of RAMs is expressed by T_n, which is related to the parallelism of the input channel of every layer. The Output Buffer Banks save the results generated from Conv Processing Engine and offer intermediate results to the Conv Computing Engine at proper time. The number of the RAMs in every Buffer Bank is T_m. The off-chip memory transfers are only needed for the input image, output prediction, and loading each layer's weights. The ping-pong buffers can avoid the overlap and improve the performance.

5. Accelerator optimization

The DSP resources in FPGA are the bottleneck of performance. To tackle the performance bottleneck and achieve a better speedup, we adopt the following optimizing strategies: pipeline, parallelization, and low precision.

5.1. Inter CNN layer pipelining architecture

The pipeline is an essential technique. This work proposes the inter layer pipeline technique to improve the throughput in depthwise

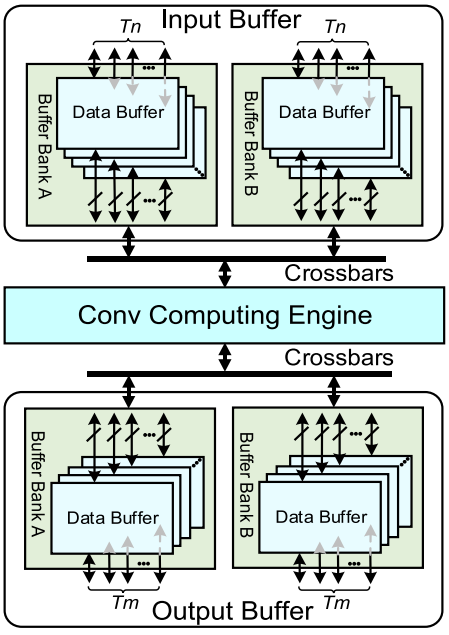

Fig. 10. Buffer structure.

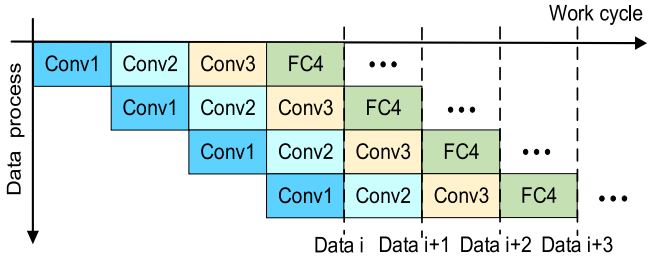

Fig. 11. Each layer works in a pipelined style.

separable CNN accelerator. As shown in Fig. 11, the Conv1-Conv3 and FC4 layers work in the pipelined style to maximize the throughput. The computational complexity of the layers is different, resulting in different operating time. To achieve a balanced pipeline, we use more resources (DSPs, RAMs, logic blocks) for convolutional layers to increase the pipeline efficiency. The goal is to achieve approximatively the same running time in each pipeline state. In the pipelined structure, the ping-pong operation is used to make the hardware work continuously.

5.2. Parallelizing computing engines

Most of the data in CNN is independent, which provides the potential for parallelism. This work adopts 3 types of parallelism to accelerate the CNN layers as proposed in [31].

(1) Parallelizing computation of multiple output feature maps

The output feature maps in the depthwise separable convolutional layers are totally independent for each other and can be calculated in parallel. As shown in Fig. 5, the output feature maps in the depthwise convolution are composed of p_n DCUs in parallel, and each DCU performs the calculation of one channel of output feature maps. Like to the depthwise convolution, the output feature maps in pointwise convolution are composed of p_m PCUs, and each PCU performs one output channel calculation. In CNN, because of the difference of computation and the number of channels of the feature maps for every layer, the parallelization parameter p_n and p_m for each layer is configured differently. Besides, we also adopt input data reuse in the design as shown in Fig. 12.

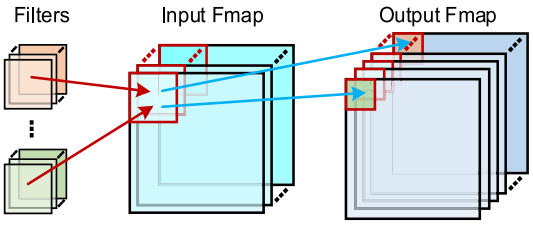

Fig. 12. Input data reuse and parallelizing computation of multiple output feature maps.

Table 3
Parallelization parameter configuration for DS-CNN implementation.

	Conv1	Conv2		Conv3		
		DW_ Conv	PW_Conv	DW_Conv	PW_ Conv	FC4
P_n	3	1	16	1	32	8
P_m	16	16	8	32	8	10

Multiple filters are applied to the same feature map, so the input feature map activations are used multiple times across filters.

(2) Parallelizing computation of multiple input feature maps

Each channel of the output feature maps is performed by the convolution operation of multiple channels of the input feature maps. Thus, the multiple input feature maps can be calculated in parallel. The parallelization parameter between the input channels of feature maps is represented by p_n. For depthwise convolution, the channels of input feature maps can be parallelized because there is no fusion of the output channels. As shown in Fig. 5, multiple DCUs in parallel constitute the depthwise convolution. For pointwise convolution, the input feature maps are performed via 1×1 convolution with a small computation, so it is possible to perform a high degree of parallel computation.

As mentioned above, the parallelization parameter of input feature maps is p_n, and the parallelization parameter of output feature maps is p_m. Considering the factors such as running time balance, efficiency of pipeline, hardware resources, the optimal parallelization parameters configuration for DS-CNN implementation are shown in Table 3. DW_Conv and DW_Conv mean depthwise convolution and pointwise convolution respectively. The convolution layer 2 (Conv2) is depthwise separable convolution with 16 channels in the input feature maps and 32 channels in the output feature maps. The depthwise convolution have no the channel fusion and one input channel corresponds to one output channel. Therefore, the parallelization parameter of input feature maps p_n is 1. The 16 channels of output feature map are performed with 16 convolutional kernels in parallel, so the parallelization parameter of output feature maps p_m is 16. The parallelization parameter configurations for other layers are similar.

(3) Parallelizing execution of convolutional kernels

Each pixel in the output feature is the result of a set of convolutions. A convolution is essentially a set of multiplications and additions. As each multiplication and addition between a pixel in an input feature map and a convolutional kernel are independent completely, so they can be performed in parallel with the configurable line buffer and MAC unit in Fig. 6. For 3×3 convolution, we can get one result of output feature map every cycle with a parallel style and the speed can be increased by 9 times.

5.3. Low precision numeric computing

The training and inference of deep neural networks with low-precision fixed-point arithmetic instead of floating-point arithmetic has been proved to be effective [37,38]. Using low-precision fixed-point

Table 4
Resource utilization of DS-CNN on Arria 10 FPGA.

Type	ALMs	Registers	RAMs(bit)	DSPs
Conv1	8088	10,982	322,199	240
Conv2	3724	12,955	461,516	144
Conv3	7304	25,609	704,839	288
FC4	878	3022	327,680	40
Utilization	19,994	52,568	1816,234	712
Available	427,200	1708,800	55,971,840	1518
Percent (%)	4.68	3.08	3.24	46.90

arithmetic is a hardware-friendly way of approximating CNNs. Work [39] shows that a 16-bit fixed-point multiplication consumes 6.2× less energy than a 32-bit floating-point, while the results is less than 0.5% loss of prediction accuracy. In this paper, we use 16-bit fixed-point arithmetic for DS-CNN. Using 16-bit fixed-point arithmetic results in less than 1% loss of prediction accuracy: 71.8% compared with 72.6% using 32-bit floating point arithmetic. However, with 8-bit fixed-point, the accuracy drops to only 61%. Although 8-bit fixed-point multiplication can reduce more energy consumption, the prediction accuracy drops too much. In this paper, each network layer is split into two groups: one for the layer output activations, one for the layer weights. The layer output activations are represented by 8-bit integer (including 1-bit sign bit) and 8-bit fraction. As weights are normally significantly smaller, the layer weights are represented by 4-bit integer (including 1-bit sign bit) and 12-bit fraction. We use the Ristretto [40] quantitative tool to approximate 32-bit floating-point DS-CNN model by condensed fixed-point model.

6. Experimental results and discussion

6.1. Experimental setup

The experiment is carried out with Intel Quartus Prime 16.0 for synthesis and ModelSim for simulation. Our implementation is built on FPGA platform: Terasic DE5a-Net-DDR4 Arria 10 GX 1150, which consists of 1150 K logic elements (LEs), 1518 DSP blocks and 53 M bits M20K RAMs. The single-precision floating-point peak performance reaches 1.5T FLOPS. We compared our design with two server-class computing platforms: Intel Xeon E5-1603@2.8 GHz multicore processor with 32G memory (CPU) and NVIDIA Titan X@1.5 GHz with 12G memory (GPU). The TensorFlow open source deep learning framework is used for software implementation.

6.2. Experimental results

In this section, we first report the resource utilization of the design on FPGA. Then, the DSP usage and execution time of every layer for STD-CNN and DS-CNN are shown. Afterwards, we compare our accelerator design on FPGA with software implementation on CPU and GPU. Finally, the comparison between our FPGA design and some other similar FPGA implications is provided.

Table 4 shows the resource utilization after synthesizing and timing for different layers of DS-CNN implemented on the Arria10 FPGA. We can tell that our CNN accelerator has utilized only 46.9% DSPs and 4.68% ALMs, which means that we have more hardware resource to expand the networks and other applications.

The STD-CNN and DS-CNN accelerator can be synthesized for a maximum clock frequency of 150 MHz and 180 MHz respectively. The difference of clock frequency for STD-CNN and DS-CNN accelerators comes from the computational complexity of the different layers and the optimizing techniques. As shown in Table 5, the execute time of each layer for STD-CNN has more imbalance than DS-CNN because of the workload of different layers. The processing time of DS-CNN for each image is less 4× than that of STD-CNN. When we compare the speed between DS-CNN and STD-CNN, we try to adopt almost the same DSPs because

Table 5
DSP usage and execution time of each layer for STD-CNN and DS-CNN.

Layer	STD-CNN		DS-CNN	
	DSPs	Time (μs)	DSPs	Tim (μs)
Conv1	240	8.34	240	6.95
Conv2	160	**64.23**	144	14.54
Conv3	320	52.25	288	**15.57**
FC4	40	14.57	40	12.15
Total	760	139.39	712	49.21

Table 6
Comparison between CPU, GPU and FPGA implementations for DS-CNN.

Platform	Xeon E5 CPU	Titan X GPU	Arria10 FPGA
Time (μs)	274	190	15.57
Performance (GOP/s)	5.62	8.10	98.91
Power(W)	60	250	8.5
Power Efficiency (GOP/s/W)	0.05	0.02	5.81
Speedup	1.0×	1.4×	17.6×
Power Ratio	7.1×	29.4×	1.0×

the computational complexity. More FPGA resources in the pipeline are employed to get balanced execution time for different layers. In our design, by adopting the pipelined structure among the layers, the processing time of each image with a size of 32 × 32 × 3 is 15.57us. The latency time of each image frame is 49.21us which is the sum of all the layers.

The execution time, performance and power of DS-CNN implementation on CPU, GPU and FPGA are summarized in Table 6. We evaluate the average execution time with 10,000 images on CPU and GPU. The test images come from CIFAR-10 dataset with a size of 32 × 32 × 3. The processing time of each image for DS-CNN (only Conv1∼ FC4) on FPGA is 15.57 μs, which is 17.6× faster than the CPU platform, 12.57× faster than GPU platform. We design specific hardware architecture e.g. pipeline fashion and intermediate result store on the on-chip memory to accelerate the DS-CNN for FPGA implementation. The DS-CNN network needs 0.77 MMACs including multiplications, adds and non-linear functions. Our system achieves an average performance of 98.91 GOP/s for the whole network, which is 12.2× higher than GPU. GPU is suitable for training with batch model while the image is processed one by one during the inference of CNN. Since the FPGA is powered only by the PCIe slot of a personal computer and the power measurement of the FPGA board itself is not straightforward. We attempt to evaluate the power consumption of the FPGA through the PowerPlay Power Analyzer Tool provided by the Quartus Prime. The power consumption is 8.5 W. Our FPGA design consumes 7.1× lower power than CPU, 29.4× lower power than GPU. Therefore, FPGA implementation uses much less energy than its software counterparts.

As depthwise separable convolution neural networks are recent proposed and developed, there is no specific FPGA-based implementation. We compare our work with some existing works on standard convolutional neural networks implementations on different FPGAs in Table 7. Besides, to compare the difference of implementation between the standard convolution and the depthwise separable convolution with same architecture on FPGA, the prototype design and optimization of STD-CNN and DS-CNN on FPGA are also provided respectively.

It may not be completely fair to compare operating time and power between a standard CNN [20,29,41] and DS-CNN, because the input image size, the computational complexity of CNN, the optimization target and FPGA platform are different. The parameter of “MOPs” in Table 7 means that one Million of Operations. One MAC operation mentioned in Section 3.1 has one multiplication and one addition. So, 1 MOPs = 0.5×10^6 MACs. To be consistent with other references, we used MOPs instead of MAC as computational complexity. As shown in Table 7, Our accelerator has a performance of 98.91 GOP/s, which outperforms the other compared FPGA accelerator design [20,34]. In ad-

Table 7
Comparison with other FPGA accelerators.

	[29]	[41]	[20]	[34]	This work	
					STD-CNN	DS-CNN
Device	Virtex5 SX240T	Zynq XC7Z045	Virtex7 VX485T	Zynq XCZU9EG	Arria10 GX 1150	Arria10 GX 1150
Frequency/MHz	120	142	100	150	150	180
Precision	48bit fixed	16bit fixed	32bit float	8bit fixed	16bit fixed	16bit fixed
DSP Consumed	N/A	900	224	1452	760	712
Power(W)	14	4	18.61	N/A	8.69	8.52
Time (μs)	32,500	23,810	21,600	7850	64.23	15.57
Complexity (MOPs)	520	552	1330	720	5.62	1.54
Performance (GOP/s)	16	23.18	61.62	91.2	87.50	98.91
Power Efficiency (GOP/s/W)	1.14	5.80	3.31	N/A	10.07	11.61
Resource Efficiency (GOP/s/DSP)	N/A	2.58E−02	2.75E−02	6.28E−02	1.15E−01	1.39E−01

dition, the proposed implementation can achieve the highest power efficiency with value 11.61 GOP/s/W in comparison with the previous designs [41]. To provide a fair comparison, we further present results of "resource efficiency" in Table 7. It is defined as average GOP/S per DSP, which can represent the efficiency of resource used for implementation on FPGA. As illustrated in the last row of Table 7, our implementation achieves the best resource efficiency, which is 2.21× better than the second best [34]. In addition, compared to STD-CNN based on standard convolution, DS-CNN based on deep separable convolution achieves higher performance, power efficiency and resource efficiency.

6.3. Discussion

At present, our FPGA-based accelerator implementation uses 16-bit fixed-point number on intel Arria 10 FPGA. The accelerator consumes 712 DSP blocks, which occupies 46.9% DSPs in total. There are some techniques to reduce the DSP consumption, such as using 8-bit fixed-point precision with better quantization strategy e.g. Non-uniform quantization, Dynamic fixed point and Reinforcement learning (RL) to automatically determine the optimal number of quantization bits for each layer [42]. The different bit allocation for every layer is a hardware-friendly approach for FPGA. Theoretically, when we adopt the 8-bit fixed-point to quantize the weight and activation value, 2×Conv Processing Engines can be placed on the same FPGA. Therefore, we can deploy the same CNN model to a lower capacity FPGA device and the DSPs, on-chip memory and power consumption can be reduced by half. In other word, the performance of Conv and FC layers can double. The architecture of depthwise separable CNN accelerator proposed in this paper is scalable. This paper maps a small scale depthwise separable convolutional neural network with 3 Conv layers and 1 FC layer into the Arria 10 FPGA. The parallelization parameters p_n of the channels of input feature maps and the parallelization parameters p_m of the channels of out feature maps can be configured differently according the CNN model size and the hardware resource.

7. Conclusions

In this work, we propose a depthwise separable convolutional neural network accelerator with all the layers working concurrently in a pipelined fashion to improve the system throughput and performance. To exploit the parallelism in CNNs, multiple hardware optimization strategies are adopted in the convolutional layer, achieving a good balance among speed, resource usage and power consumption. Besides, in the FC layer, data titling technique is adopted to divide matrix multiplication from large dimension into small dimension. The intermediate data of each layer is buffered in the on-chip memory to reduce the required memory bandwidth and data access power consumption. With the DS-CNN model as case study, experimental results show that our implementation on Arria 10 FPGA can achieve a performance of 98.91 GOP/s. To process one image with the size of 32×32×3, our design can achieve up to 17.6× speedup than CPU and 29.4× low energy than GPU respectively. More prominently, in term of energy efficiency and resource efficiency, our proposed depthwise separable CNN accelerator is superior to the standard CNN accelerator with the same computing scale. Our design and verification experiments have shown that depthwise separable CNN is an excellent candidate as an embedded deep network accelerator for mobile edge computing. We would like to extend our design for higher resolution images involved in mobile edge computing as our future work.

Acknowledgments

This work was supported by National Key Research and Development Program of China under grant 2017YFA0206104, Shanghai Municipal Science and Technology Commission under grant 18511111302, Key Projects of Bureau of International Cooperation Chinese Academy of Sciences under grant 184131KYSB20160018, National Natural Science Foundation Youth Fund under grant No. 61704179.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[2] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556, 2014.
[3] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2015, pp. 91–99.
[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: unified, real-time object detection, in: Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788.
[5] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask R-CNN, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980–2988, doi:10.1109/ICCV.2017.322.
[6] V. Badrinarayanan, A. Kendall, R. Cipolla, SegNet: a deep convolutional encoder-decoder architecture for image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 39 (12) (2017) 2481–2495.
[7] N. Tajbakhsh, J.Y. Shin, S.R. Gurudu, et al., Convolutional neural networks for medical image analysis: full training or fine tuning? IEEE Trans. Med. Imaging 35 (5) (2016) 1299–1312, doi:10.1109/TMI.2016.2535302.
[8] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: a unified embedding for face recognition and clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 815–823.
[9] A. Ucar, Y. Demir, C. Guzelis, Object recognition and detection with deep learning for autonomous driving applications, (in English), Simul.-Trans. Soc. Model. Simul. Int. 93 (9) (Sep 2017) 759–769, doi:10.1177/0037549717709932.
[10] P. Pelliccione, E. Knauss, R. Heldal, et al., Automotive architecture framework: the experience of volvo cars, J. Syst. Archit. 77 (2017) 83–100. 06/01/ 2017 https://doi.org/10.1016/j.sysarc.2017.02.005.
[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898.
[12] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.
[13] S. Han, H. Mao, and W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, arXiv:1510.00149, 2015.
[14] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, and K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size, arXiv:1602.07360, 2016.

[15] F. Chollet, Xception: deep learning with depthwise separable convolutions, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1800–1807, doi:10.1109/CVPR.2017.195.
[16] A.G. Howard, M. Zhu, B. Chen et al., Mobilenets: efficient convolutional neural networks for mobile vision applications, arXiv:1704.04861, 2017.
[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[18] L. Sifre, P. Mallat, Rigid-motion Scattering for Image Classification, Citeseer, 2014.
[19] S. Chetlur, C. Woolley, P. Vandermersch et al., cudnn: efficient primitives for deep learning, arXiv:1410.0759, 2014.
[20] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, in: Presented at the Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2015, doi:10.1145/2684746.2689060.
[21] T. Luo, S. Liu, L. Li, DaDianNao: a neural network supercomputer, IEEE Trans. Comput. 66 (1) (2017) 73–88, doi:10.1109/TC.2016.2574353.
[22] N.P. Jouppi, C. Young, N. Patil, et al., In-datacenter performance analysis of a tensor processing unit, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 1–12, doi:10.1145/3079856.3080246.
[23] J.P. Cobos Carrascosa, J.L. Ramos Mas, B. Aparicio del Moral, M. Balaguer, A.C. López Jiménez, J.C. del Toro Iniesta, SIMD architecture on FPGA for scientific computing aboard a space instrument, J. Syst. Archit. 62 (2016) 1–11. 01/01/2016 https://doi.org/10.1016/j.sysarc.2015.10.006.
[24] M. Dehnavi, M. Eshghi, FPGA based real-time on-road stereo vision system, J. Syst. Archit. 81 (2017) 32–43. vol 11/01/2017 https://doi.org/10.1016/j.sysarc.2017.10.002.
[25] K. Guo, L. Sui, J. Qiu, et al., Angel-eye: a complete design flow for mapping CNN onto embedded FPGA, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (1) (2018) 35–47, doi:10.1109/TCAD.2017.2705069.
[26] N. Suda, V. Chandra, G. Dasika, et al., Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks, in: Presented at the Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2016, doi:10.1145/2847263.2847276.
[27] L. Huimin, F. Xitian, J. Li, C. Wei, Z. Xuegong, W. Lingli, A high performance FPGA-based accelerator for large-scale convolutional neural networks, in: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016, pp. 1–9, doi:10.1109/FPL.2016.7577308.
[28] M. Sankaradas, V. Jakkula, S. Cadambi, et al., A massively parallel coprocessor for convolutional neural networks, in: 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009, pp. 53–60, doi:10.1109/ASAP.2009.25.
[29] S. Chakradhar, M. Sankaradas, V. Jakkula, S. Cadambi, A dynamically configurable coprocessor for convolutional neural networks, SIGARCH Comput, Archit. News 38 (3) (2010) 247–257, doi:10.1145/1816038.1815993.
[30] Y. Ma, Y. Cao, S. Vrudhula, J.-s. Seo, Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks, in: Presented at the Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, California, USA, 2017, doi:10.1145/3020078.3021736.
[31] M. Motamedi, P. Gysel, V. Akella, S. Ghiasi, Design space exploration of FPGA-based deep convolutional neural networks, in: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 575–580, doi:10.1109/ASP-DAC.2016.7428073.
[32] A. Rahman, S. Oh, J. Lee, K. Choi, Design space exploration of FPGA accelerators for convolutional neural networks, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, 2017, pp. 1147–1152, doi:10.23919/DATE.2017.7927162.
[33] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, J. Cong, Energy-efficient CNN implementation on a deeply pipelined FPGA cluster, in: Presented at the Proceedings of the 2016 International Symposium on Low Power Electronics and Design, San Francisco Airport, CA, USA, 2016, doi:10.1145/2934583.2934644.
[34] J. Su, J. Faraone, J. Liu, et al., Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet Classification, in: Applied Reconfigurable Computing. Architectures, Tools, and Applications, Cham, Springer International Publishing, 2018, pp. 16–28.
[35] Y. LeCun, C. Cortes, C. Burges, MNIST Handwritten Digit Database, 2, AT&T Labs [Online], 2010 Available http://yann.lecun.com/exdb/mnist.
[36] A. Krizhevsky, V. Nair, G. Hinton, The CIFAR-10 dataset, 2014. online http://www.cs.toronto.edu/~kriz/cifar.html.
[37] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, Quantized neural networks: training neural networks with low precision weights and activations, arXiv:1609.07061, 2016.
[38] J. Wu, C. Leng, Y. Wang, Q. Hu, J. Cheng, Quantized convolutional neural networks for mobile devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4820–4828.
[39] S. Han, X. Liu, H. Mao, et al., EIE: efficient inference engine on compressed deep neural network, in: Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on, IEEE, 2016, pp. 243–254.
[40] P. Gysel, J. Pimentel, M. Motamedi, S. Ghiasi, Ristretto: a framework for empirical study of resource-efficient inference in convolutional neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2018) 1–6, doi:10.1109/TNNLS.2018.2808319.
[41] V. Gokhale, J. Jin, A. Dundar, B. Martini, E. Culurciello, A 240G-ops/s mobile coprocessor for deep neural networks, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 696–701, doi:10.1109/CVPRW.2014.106.
[42] J. Wu, Y. Zhang, H. Bai, et al., PocketFlow: an automated framework for compressing and accelerating deep neural networks, Presented at the Advances in Neural Information Processing Systems (NIPS), Workshop on Compact Deep Neural Networks with Industrial Applications, 2018.

Wei Ding received the B.S. degree in microelectronics science and engineering from School of Physics and Technology, Wuhan University, Wuhan, China, in 2016. He is currently pursuing a M.S. degree in electronic and communication engineering at University of Chinese Academy of Sciences, Beijing 100049, China. His main research is on the deep learning hardware acceleration and digital integrated circuit design.

Zeyu Huang received the B.S. degree in electronics and communication engineering from University of Jinan, Jinan, China, in 2016. He is working toward his M.S. degree from University of Chinses Academy of Sciences, Beijing 100049, China. His major is electronic and communication engineering. His recent research interest is in the picture processing of multi-camera and digital integrated circuit design.

Zunkai Huang received the B.S. degree in electronics engineering from Tianjin University, Tianjin, China, in 2013, and the Ph.D. degree in microelectronics from Shanghai Advanced Research Institute, Chinese Academy of Sciences, Shanghai, China, in 2018. He was with the Hiroshima University, Japan, as a special research student, from 2016 to 2017. Since July 2018, he has been with the Shanghai Advanced Research Institute, Chinese Academy of Sciences, where he is currently an assistant professor. His research focuses on CMOS image sensor chip and system, digital image signal processing circuits, and driving circuits for display panels.

Li Tian received the Ph.D. degree in electronics science and technology from Shanghai Institute of Technical Physics of the Chinese Academy of Sciences in 2013. He is currently working as an associate professor at Shanghai Advanced Research Institute, Chinese Academy of Sciences. His research focuses on CMOS image sensor chip, smart vehicle vision systems and digital image signal processing circuits and algorithms.

Hui Wang received the Ph.D. degree in Physics from the Institute of Semiconductors, Chinese Academy of Sciences, Beijing, China, in 2001. He had a postdoctoral position at IMEC, Belgium, and then worked as an Associate Professor at Shanghai Jiao Tong University, Shanghai, China. In spring 2010, he joined the Shanghai Advanced Research Institute, Chinese Academy of Sciences, as a full professor in Microelectronics. His research interests include high-performance imaging and display panel driving.

Songlin Feng received the B.S. degree in physics from Wuhan University, Wuhan, China, in 1983, and the Ph.D. degree in semiconductor physics from Paris University, Paris, France, in 1998. After graduation, he worked in Semiconductor Institute, Chinese Academy of Sciences, Beijing, China. In 2001, he transferred to Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, where he had served as a Professor and the Director of the Institute. In 2008, he began to work in Shanghai Advanced Research Institute, Chinese Academy of Sciences, as the Director of the Institute. His current research interests are in the fields of wireless sensor network and microsystem technologies. He has published more than 120 papers in international journals.