

TFE: Energy-efficient Transferred Filter-based Engine to Compress and Accelerate Convolutional Neural Networks

Huiyu Mo
Tsinghua University
mohy15@mails.tsinghua.edu.cn

Leibo Liu
Tsinghua University
liulb@tsinghua.edu.cn

Wenjing Hu, Wenping Zhu
Tsinghua University
{huwj, zhuwp}@mail.tsinghua.edu.cn

Qiang Li
Intel Corporation
eric.q.li@intel.com

Ang Li
Tsinghua University
lia19@mails.tsinghua.edu.cn

Shouyi Yin
Tsinghua University
yinsy@tsinghua.edu.cn

Jian Chen, Xiaowei Jiang
Alibaba Group
{j.chen, xiaowei.jx}@alibaba-inc.com

Shaojun Wei
Tsinghua University
wsj@tsinghua.edu.cn

Abstract—Although convolutional neural network (CNN) models have greatly enhanced the development of many fields, the untenable number of parameters and computations in these models yield significant performance and energy challenges in hardware implementations. Transferred filter-based methods, as very promising techniques that have not yet been explored in the architecture domain, can substantially compress CNN models. However, their straightforward hardware implementation inherently incurs massive redundant computations, causing significant energy and time consumption. In this work, a highly efficient transferred filter-based engine (TFE) is developed to alleviate this deficiency, with CNN models compressed and accelerated. First, the filters of CNN models are flexibly transferred according to specific tasks to reduce the model size. Then, two hardware-friendly mechanisms are proposed in the TFE to remove duplicate computations caused by transferred filters, which can further accelerate transferred CNN models. The first mechanism exploits the shared weights hidden in each row of transferred filters and reuses the corresponding same partial sums, reducing at least 25% of repetitive computations in each row. The second mechanism can intelligently schedule and access the memory system to reuse the repetitive partial sums among different rows of the transferred filters with at least 25% of computations eliminated. Furthermore, an efficient hardware architecture is proposed in the TFE to fully reap the benefits of the two proposed mechanisms such that different types of networks are flexibly supported. To achieve high energy efficiency, the sub-array-based filter mapping method (SAFM) is proposed, where the process element (PE) sub-array is used as the elementary computational unit to support various filters. Therein, input data can be efficiently broadcast in each PE sub-array and the load can be stripped from each PE and intensively alleviated, which can dramatically reduce the area and power consumption. Excluding MobileNet-like networks that adopt depth-wise convolution, most mainstream networks can be compressed and accelerated by the proposed TFE. Two state-of-the-art transferred filter-based methods, i.e., doubly CNN and symmetry CNN are implemented by exploiting the TFE. Compared with Eyeriss, average speedup improvements of 2.93× and 3.17× are achieved in the convolutional layers of various modern CNNs. The overall energy efficiency can be improved by 12.66× and 13.31× on average. Compared with other state-of-the-art related works, the TFE can maximally achieve a parameter reduction of 4.0×, a speedup of 2.72× and an energy efficiency improvement of 10.74× on VGGNet.

I. INTRODUCTION

Recently, deep convolutional neural networks (CNNs) have achieved great success in computer vision [49], natural language processing [31], speech recognition [52], etc. However, these works largely rely on millions of parameters and computations to achieve high accuracy. A large number of parameters have to be stored in off-chip DRAM, which causes frequent access to the off-chip DRAM [6], [37]. In addition, the large amount of computations require too many hardware resources to maintain an acceptable running speed, which incurs an intolerable area and power overhead [5], [7], [34].

To alleviate these problems, a number of studies have been undertaken to compress CNN models or accelerate CNN implementations. Some methods exploit the property of the convolutional (CONV) process to reduce inconsequential CONV computations, such as prediction-based methods [2], [45] and Winograd algorithm-based methods [32], [53], herein called computation-reduction-based methods. However, the CNN model size is still kept uncompressed in these methods. In contrast, some methods make use of weight sparsity and redundancy to compress the model size, such as weight pruning and weight quantization [10], [16], [18], [29], which are referred to as weight-compression-based methods in this work. Based on these methods, many architectures [11], [16], [27], [37], [51] have been further designed to implement computations of nonzero weights and activations to accelerate the CNN process. However, as nonzero activations and weights are both irregularly distributed, eliminating zero-related computations can be very difficult and complex, which can seriously degrade the final energy efficiency improvement [5], [7], [19]. Recently, transferred filter-based algorithms, as a kind of promising and attractive network model compression method, have become increasingly popular and used in many machine learning fields, such as face alignment [33], face detection [57], natural language processing [46] and other network compression and pruning studies [13], [43]. The trans-

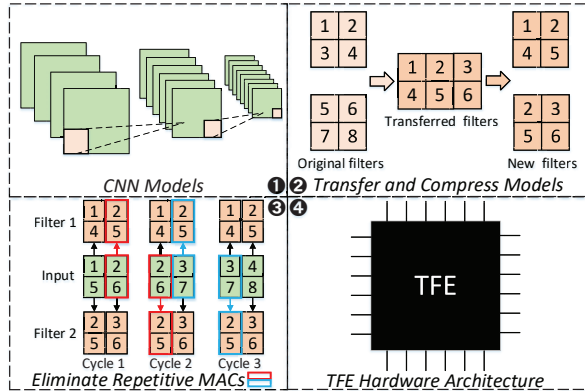


Fig. 1. TFE process from step 1 to step 4.

ferred filter-based algorithms exploit the potential redundancy property of CNN models and change the structure of filters to largely reduce network parameters with an acceptable accuracy loss [7], [8]. Though a high compression rate is achieved in transferred filter-based algorithms, the common weights in different transferred filters will be convolved with the same input feature maps (ifmaps) to generate different output feature maps (ofmaps). Therefore, massive redundant computations are incurred, which causes significant energy consumption.

In this work, a transferred filter-based engine (TFE) is proposed to compress the CNN model size and accelerate the compressed CNN models in a hardware-friendly manner without large overhead. As shown in Fig. 1, first, the CNN models are initially ready for the process. Second, according to different tasks, specific transferred filters are applied to CNN models to compress the model size in the TFE. Third, the repetitive multiply-accumulate (MAC) operations in compressed networks are eliminated by the proposed techniques. Finally, the TFE hardware architecture can efficiently accelerate these networks with less area and power consumption.

In the TFE, the input activation and transferred filters are convolved row by row in the process element (PE) array. To eliminate repetitive computation inside each transferred filter row because of shared weights, we propose an efficient technique, product and partial sum reuse (PPSR), where the intermediate products and partial sums (PSums) from the PE array are stored in the stacked registers (SRs) group. These values can be directly reused or added to produce the row results of different transferred filters. Additionally, there are repetitive weights among different transferred filter rows, which produce repetitive entire-row CONV results among different transferred filters. In contrast to the conventional memory access strategy, entire-row result reuse (ERRR) is proposed to eliminate these redundant computations. The entire-row CONV results between each input row and all transferred filter rows are stored in different memory locations. Each time, the results of each row in the memory are read to obtain CONV window results of different transferred filters. Once the related computations are finished, they will be overwritten by the results of another row. Therefore, these memories can be cyclically written and read to save computations and reduce

the memory requirement.

To fully reap the benefits of the above hardware-friendly techniques, we further develop a highly flexible hardware architecture to support different types of networks in TFE. To achieve high energy efficiency, the sub-array-based filter mapping method (SAFM) is proposed in TFE hardware architecture, which uses a static 3×3 and 4×4 PE sub-array rather than a single PE as the basic computational unit to support various filter mappings. Additionally, the input data is broadcasted in each PE sub-array, and the corresponding products are then simultaneously obtained. The experimental results show that SAFM can significantly reduce area and power consumptions. Furthermore, the products corresponding to the same ofmaps are first accumulated and then written into registers, which are 85.9% less than conventional designs.

To evaluate the effectiveness of the proposed TFE, several state-of-the-art CNNs are used as benchmarks for evaluation, including AlexNet [26], VGGNet [42], ResNet (ResNet-56 is used here) [17] and GoogLeNet [47]. Moreover, three recent networks, DenseNet [21], SqueezeNet [22] and Residual Attention Network (ResANet) [50], are also evaluated to further prove the efficiency of the TFE. First, these networks are converted to transferred filter-based networks (e.g., DCNN and SCNN) while retaining similar accuracy. Then, we compare our proposed TFE with other modern architectures for the transferred networks. For example, when compared with Eye-riss [6], average speedups of $2.73 \times$ (6×6 DCNN) and $2.97 \times$ (SCNN) are achieved for all networks including fully connected (FC) layers. Moreover, the overall off-chip energy access can be reduced by $1.46 \times$ (6×6 DCNN) and $1.48 \times$ (SCNN). Attributed to the TFE, an average of $12.66 \times$ (6×6 DCNN) and $13.31 \times$ (SCNN) energy efficiency improvements are achieved on VGGNet and AlexNet. Furthermore, we compare the proposed TFE with other similar works, including weight-compression-based and computation-reduction-based methods. The experimental results show that the TFE can achieve a parameter reduction rate and a speedup comparable to those of weight-compression-based methods [16], [19], [37], [51] with more hardware-friendly CONV operations. Meanwhile, at almost the same accuracy loss ($\leq 1\%$) on ImageNet [12], $2.56 \times$ speedup, $4.0 \times$ parameter reduction and $8.99 \times$ energy efficiency improvements on average are achieved by the TFE (SCNN) on AlexNet and VGGNet compared with SnaPEA [2], which demonstrates the high efficiency of TFE.

The transferred filter-based algorithms in the TFE focus on reducing redundancy among weights in different filters with the same input feature map in the canonical convolution (including dilated convolution and deconvolution). Therefore, the transferred filter-based algorithms in the TFE significantly benefit most mainstream networks, including computation-intensive networks (e.g., VGGNet [42]), edge networks (e.g., SqueezeNet [22]) and depth-based networks (e.g., ResNet [17]). In contrast, the TFE is not beneficial to MobileNet [20] and other similar compact networks, which adopt depth-wise convolution to eliminate the overlap of 3×3 filters and substantially remove the redundancy. However, MobileNet-like

compact networks have large accuracy gaps compared with mainstream complex networks. Meanwhile, 1×1 filters are the most simplest filters and cannot be further transferred through translation and rotation. Because common transformations in transferred filter-based algorithms, such as translation [55], symmetry [9] and rotation [13], are friendly to software and hardware implementation with marginal accuracy loss ($\downarrow 1\%$), it is preferable for most networks to be transferred and accelerated in the TFE.

In summary, the TFE makes the following contributions to compressing and accelerating CNN models:

- The TFE is the first systematic solution that exploits transferred filter-based algorithms to significantly compress and accelerate CNN models.
- Two hardware-friendly techniques, PPSR and ERRR, are proposed in the TFE to eliminate hidden repetitive computations inside and among transferred filter rows. Therefore, the merging of the two techniques can remove all duplicate computations in transferred filters, significantly accelerating CNN implementations.
- An efficient TFE hardware architecture is developed to use SAFM to efficiently support various filters. SAFM adopts the PE sub-array as the basic computational unit where input data can be efficiently broadcast, which can substantially decrease the area overhead and power consumption of the CNN implementation.

II. BACKGROUND AND MOTIVATION

Recently, transferred filter-based algorithms, promising network compression methods, have become increasingly attractive and popularly used in many machine learning applications. There are mainly four state-of-the-art transferred filter-based algorithms, namely doubly CNN (DCNN) [55], symmetry CNN (SCNN) [9], concatenated rectified linear unit (CReLU) [38] and multi-bias nonlinear activation (MBA) [30]. The roles of the four algorithms acting in various tasks are different but all can be applied to most networks [5], [7], [34]. Among the four state-of-the-art transferred filter-based algorithms, CReLU [38] trains pairwise positive-negative effective filters to extract both positive and negative phase information of an input signal. MBA [30] assigns multiple biases to input features to enhance the weak signal of filters. These two transferred filter-based algorithms can both compress the network size. However, they are implemented on the conventional CNN architecture through specific control logic. Therefore, to highlight the advantages of the proposed architecture, DCNN and SCNN with better generality [9], [13], [43], [55] are mainly considered and analyzed in this work.

A. Transferred Filters

DCNN rests on the finding that many filters in well-trained CNNs are slightly translated versions of each other. Based on this observation, the DCNN learns a group of filters that are the translated versions of each other. As shown in Fig. 2(a), the filters that are convolved with the same feature are trained as a set of the meta filters that has larger filter sizes than the

TABLE I
SHAPE PARAMETERS OF A TRANSFERRED CNN LAYER.

Shape Parameter	Description
N	# of ifmap channels/filter channels
M	# of ofmap channels/filters
H/W	ifmap height/width
E/F	ofmap height/width
K/K	transferred filter height/width
Z/Z	meta filter height/width

original effective filter size. Transferred filters (the same size as the original filter) then can be extracted from each meta filter and convolved with the input. The corresponding shape parameters are listed in Table I. Without loss of generality, 4×4 and 6×6 meta filter sizes are used in the DCNN to evaluate the effects of the proposed techniques, which can minimize the classification accuracy loss while maximizing computation and parameter reductions.

SCNN learns that filters in CNN models always detect particular patterns at multiple spatial locations in the input and these patterns often occur in different orientations. Therefore, multiple copies of the same filter in different orientations will often exist. To make the CNN models more compact and avoid overfitting, a form of rotational symmetry, including rotation by a step of 90° rotation and horizontal/vertical flipping, is applied to the original filters as shown in Fig. 2(b). Likewise, symmetric effective filters are convolved with the same ifmaps.

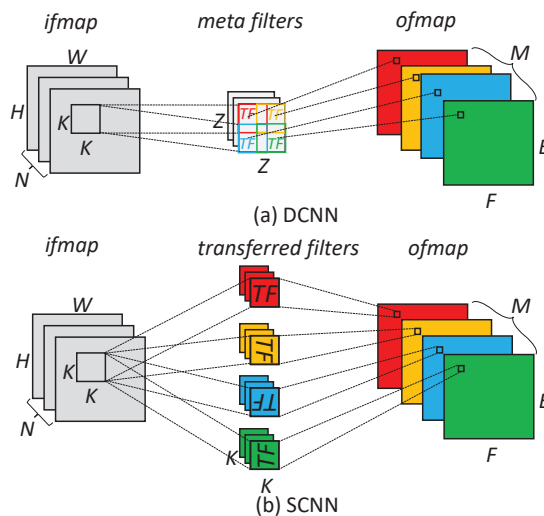


Fig. 2. Transferred filters, (a) DCNN and (b) SCNN. "TF": transferred filter.

B. Repetitive Computation Analysis in Transferred Filters

Though transferred filter-based algorithms are used in the TFE to compress CNN models, massive repetitive computations arise, which prejudice the CNN acceleration.

The meta filter of the DCNN is partitioned into several transferred filters that implement the CONV process, similar to the original network. Taking the horizontal CONV process between one input row and one meta filter row as an example, as shown in Fig. 3(a), one row of the meta filter $\langle w_0, w_1, w_2, w_3 \rangle$

is first partitioned into two transferred filters $\langle w_0, w_1, w_2 \rangle$ and $\langle w_1, w_2, w_3 \rangle$. Then, one input row $\langle a_0, a_1, a_2 \rangle$ is convolved with the two transferred filters. Because of the translation property in the meta filter, there are repetitive weights in the two transferred filters. Therefore, input values will repeat the multiplication operation with the same weight. As shown in Fig. 3(a), $\langle a_0, a_1, a_2 \rangle$ are multiplied with the same weights $\langle w_1 \rangle$ (red dashed box) and $\langle w_2 \rangle$ (green dashed box) in the two transferred filters, causing two repetitive computations.

In the SCNN, some transferred filters are horizontally symmetric. As shown in Fig. 3(b), two horizontally symmetric filter rows $\langle w_0, w_1, w_2 \rangle$ and $\langle w_2, w_1, w_0 \rangle$ are convolved with one input row $\langle a_0, a_1, a_2 \rangle$. Because of the horizontal symmetric property, $\langle a_0, a_1, a_2 \rangle$ are also multiplied with the same weights $\langle w_1 \rangle$ (red dashed box) and $\langle w_2 \rangle$ (green dashed box) in the two symmetric filters, causing two repeated computations. As the size of the input row is much larger than the transferred filter size, the repetitive computations are substantial.

Additionally, there are repetitive filter rows in the vertical direction of transferred filters in the DCNN (Fig. 4(a)) and SCNN (Fig. 4(b)). When these repetitive filter rows are convolved with the same ifmaps, certain amounts of repetitive computations are caused in different transferred filters.

III. REPETITIVE COMPUTATION OPTIMIZATION

According to the above analysis, if shared products and PSum of different filters are temporarily stored, and then reused to produce activations of different ofmaps, redundant operations can be eliminated, and the CNN implementation can be greatly accelerated in TFE. To achieve this goal, two techniques, PPSR and ERRR, are proposed in this section.

A. Shared Partial Sums and Products

As repetitive weights cause redundant computations, the weight stationary (WS) dataflow [9], [24] is normally used here to avoid duplicated read operations of these weights.

To clearly show the shared PSum and products in the dataflow, the meta filter size is assumed as 4×4 and one row (4×1) is taken as an example. The WS dataflow of the

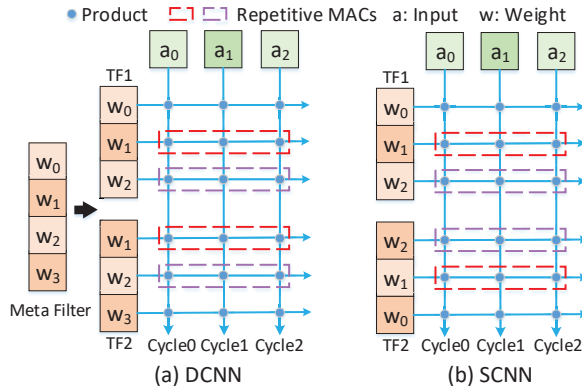


Fig. 3. Repetitive computations of the CONV process between one input row and two transferred filter rows in (a) the DCNN and (b) the SCNN.

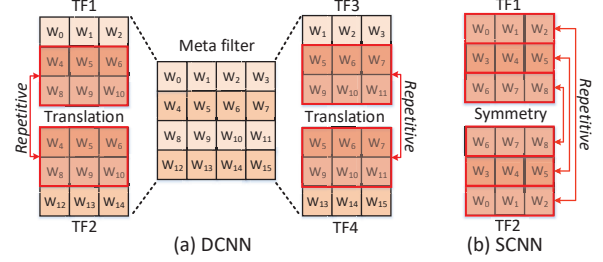


Fig. 4. Repetitive transferred rows in (a) the DCNN and (b) the SCNN.

DCNN is shown in Fig. 5(a). Each weight $\langle w_0, w_1, w_2, w_3 \rangle$ is stored in independent PEs, and one row of input data, $\langle a_0, a_1, a_2, a_3, a_4 \rangle$, is broadcast to all PEs one by one. At cycles 1 and 2, $a_1 w_1$ and $a_2 w_2$ are generated, respectively, and their addition to $a_0 w_0$ produces PSum for the first transferred filter $\langle w_0, w_1, w_2 \rangle$. Furthermore, $a_1 w_1$ and $a_2 w_2$ are also required to be summed with $a_3 \times w_3$ in the second transferred filter $\langle w_1, w_2, w_3 \rangle$ at cycle 3. The same operation also occurs in the subsequent CONV process. In the conventional WS dataflow, $a_1 w_1$ and $a_2 w_2$ will be recomputed, causing redundant computations. Additionally, we can observe that $a_1 w_1$ and $a_0 w_0$ are first added, and then, $a_2 w_2$ is further added to their sum. However, $a_1 w_1$ will be independently summed with $a_2 w_2$ to generate PSum for the second transferred filter. Therefore, in addition to the sum of $a_1 w_1$ and $a_2 w_2$, the product $a_1 w_1$ should not be overwritten immediately.

The WS dataflow of the SCNN is shown in Fig. 5(b). Likewise, one row of horizontally symmetric transferred filters, $\langle w_0, w_1, w_2 \rangle$ and $\langle w_2, w_1, w_0 \rangle$, is taken as an example here. At cycles 0 and 1, $a_0 w_0$ and $a_1 w_1$ are added to the first transferred filter, while $a_0 w_2$ and $a_1 w_1$ must be added to the second filter, namely the product $a_1 w_1$ is shared by the two transferred filters. Then at cycle 2, $a_2 w_0$ and $a_2 w_2$ are summed with the above PSum (red dashed boxes in Fig. 5(b)). Furthermore, $a_2 w_0$ and $a_2 w_2$ are used in the subsequent CONV process of two transferred filters (brown dashed boxes). However, these products will be recomputed in the conventional WS dataflow.

According to the above analysis, many products and partial sums can be shared in the transferred filter-based methods. In the above example, 33.3% and 50.0% of repetitive computations occur in the DCNN and SCNN, respectively. However, conventional WS dataflow cannot make use of this property.

B. Product and Partial Sum Reuse

The PPSR method is proposed to reduce repetitive computations and execution time. In PPSR, SRs are first designed to hierarchically store single products and PSum that will be reused in subsequent cycles. Second, specific dataflows are proposed to reuse data in SRs to produce effective filter results in different transferred filters. As 3×3 filters and 4×4 meta filters are widely used in most networks, they are used as the basic filter to illustrate the proposed idea throughout the paper.

In the DCNN, one meta filter row $\langle w_0, w_1, w_2, w_3 \rangle$ and one input data row $\langle a_0, a_1, a_2, a_3 \rangle$ are presented. First, each weight is pre-stored in independent PEs. Then, as shown in Fig. 6, at

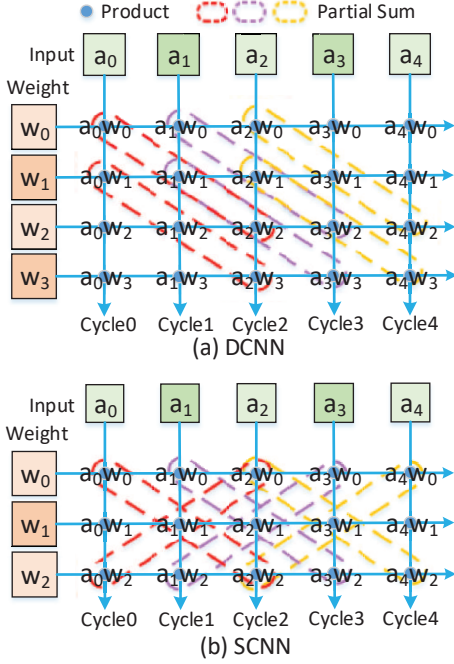


Fig. 5. Shared partial sums and products in (a) the DCNN and (b) the SCNN.

cycle 0, the input data a_0 is broadcast and multiplied with each weight. The corresponding products ($a_0w_0, a_0w_1, a_0w_2, a_0w_3$, denoted a_0 products) are stored in the bottom register of the SR. At cycle 1, the input data a_1 is broadcast and multiplied with each weight. Next, the a_0 products are transferred to right-neighbor SRs and summed with the a_1 products. The sums are stored in the second register from the bottom and the a_1 products overwrite the a_0 products. At cycle 2, these sums are transferred to the right-neighbor SRs and summed with the a_2 products. The results (red rectangle) are stored in the third register from the bottom and written into memory at the next cycles. Moreover, similar to the a_0 products, a_1 products are also transferred to the right-neighbor SRs and summed with the a_2 products. The sums are stored in the second register from the bottom and cover the a_0 sums. The above process is implemented regularly for the subsequent input data. Therefore, in the subsequent regular process, two PSums, R_{kl}^{ij} and R_{kl}^{ij+1} , are produced by each 4×1 meta filter row at each cycle, where i is the input row index, k is the meta filter row index, j is the transferred filter index and l is the PSum index in the i input row. In this way, the shared PSums and products can be reused and repetitive computations in the horizontal CONV process of the meta filters can be eliminated. Moreover, the results from transferred filters in one meta filter can be output simultaneously. Therefore, computations and execution time in the DCNN process can be significantly reduced.

In the SCNN, PPSR is used to exploit its property of repetitive weight distribution in each row, while the repetitive weight distribution property among different transferred filter rows of the SCNN can be exploited by the other proposed technique. As shown in Fig. 7, in the first input row, the

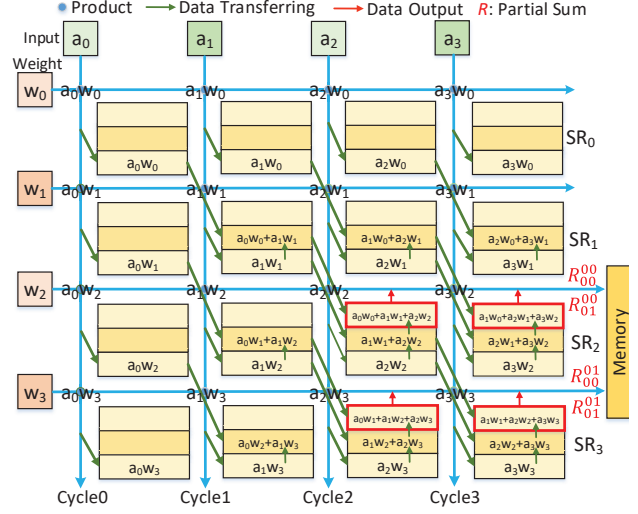


Fig. 6. PPSR used in the DCNN. R_{kl}^{ij} is the l th PSum produced by the k th row of the j th transferred filter and the i th row input data.

a_0 products are generated and summed with the a_1 product. Different from the DCNN, both a_0w_0 and a_0w_2 need to be summed with a_1w_1 and the sums are stored in SR_1 . At cycle 2, the two sums are transferred to the right-neighbor and left-neighbor SRs and summed with the respective a_2 product. Their PSums (red rectangle) are produced for transferred filters that are horizontally symmetric with each other. Additionally, the sums of a_1 products and a_2 products ($a_1w_0 + a_2w_1$ and $a_1w_2 + a_2w_1$) are stored in SR_1 and then reused for summing with other products (a_3w_2 and a_3w_0) to form PSums in the subsequent CONV window at cycle 3. In this way, the symmetric property of the SCNN in the horizontal direction, as shown in Fig. 5(b), is adequately exploited. The shared PSums (mainly stored in SR_1) are reused, which can not only eliminate repetitive computations but also reduce the execution time, as multiple results can be obtained at each cycle. Note that when the mode of the conventional CONV process is being implemented, no data will be transferred to left-neighbor SRs ($SR_2 \rightarrow SR_1 \rightarrow SR_0$).

To summarize, SR group is used in PPSR to store every PSum or product that can be reused. Then, dedicated dataflows are proposed to exploit the property of transferred filters and reuse these PSums and products, which largely eliminates redundant computations and accelerate CONV processes.

C. Entire Row Result Reuse

Instead of a tile-manner CONV process, we output activations row by row. Namely, in PPSR, the entire row CONV results between one input row and all filter rows are stored in the same memory. At each cycle, one input data instance can generate 8 PSums (corresponding to 8 transferred filters) in the 4×4 DCNN and 6 PSums (corresponding to 6 transferred filters) in the SCNN. Therein, each 2 PSums produced by one filter row are packed as a group of data stored in one memory bank each time. In the conventional architecture, two

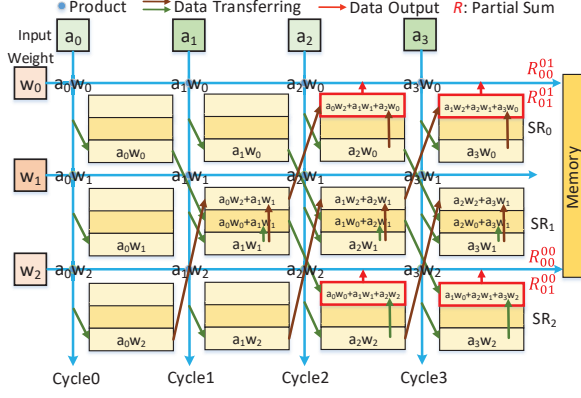


Fig. 7. PPSR used in the SCNN. R_{kl}^{ij} is the l th PSum produced by the k th row of the j th transferred filters and the i th row input data.

global memories are used to store the intermediate data of the CONV and FC layers. One provides input data to the arrays and the other stores the output data from the PE array. Once new PSums are output from the PE array, previous PSums in memory are overwritten. However, these PSums can actually be used for different transferred filters as shown in Fig. 4, which cannot be exploited by the conventional memory system. To address the above problems, ERRR is proposed to use several memories and cyclically reuse data in these memories. To clearly present ERRR, the 3×3 CONV process is taken as an example in this section.

In the 3×3 CONV process, data from three input rows are required to produce the results in CONV windows. In PPSR, each input row is convolved with four rows of the meta filter in the DCNN, and the results of different input rows are stored in different memory locations. Without loss of generality, we select only one unit of all banks in each memory as an example to describe ERRR clearly. As shown in Fig. 8, the first input row (a_0, a_1, a_2) is convolved with four meta filter rows. Each meta filter row will produce two PSums at each cycle, namely $\langle R_{00}^{00} R_{01}^{00} \rangle$, $\langle R_{10}^{00} R_{11}^{00} \rangle$, $\langle R_{20}^{00} R_{21}^{00} \rangle$ and $\langle R_{30}^{00} R_{31}^{00} \rangle$ for four meta filter rows. The 4 groups of PSums are stored in 4 banks of MEM0. Likewise, 8 PSums derived from the next input row and four meta filter rows are stored in 4 banks of MEM1.

When the third input row is convolved with the meta filter, the PSums are written into MEM2. Furthermore, the PSums in the 2 banks of MEM0 and MEM1 are read out and summed with the third input row's PSums to generate the result of a complete CONV window. Because one meta filter contains four transferred filters, the results of four ofmaps can be obtained each time. As shown in Fig. 8, the PSums in the green rectangle are added, corresponding to the two transferred filters in the upper meta filter (green rectangle). The PSums in the red rectangle are added, corresponding to the two transferred filters in the lower meta filter (red rectangle). Therefore, MEM0 and MEM1 are both in the read state, while MEM2 is in the write state for a short time, called period 0. In period 1, the fourth input row is processed with the meta filter, and the PSums need to be summed with the PSums in MEM1 and MEM2

to generate the results in the CONV windows of the next row for the same four ofmaps. Some PSums in MEM1 and MEM2 can be reused while the PSums in MEM0 will no longer be used. Therefore, MEM0 can be reused to store results of the fourth input row for the subsequent CONV process. At this time, MEM1 and MEM2 are both in the read state while MEM0 is in the write state. Likewise, in period 2, the fifth input row is processed, and MEM1 is reused to store the corresponding results. Thus, both the memory and PSums of the entire row are cyclically reused, saving hardware resources and eliminating repetitive MACs among different filter rows.

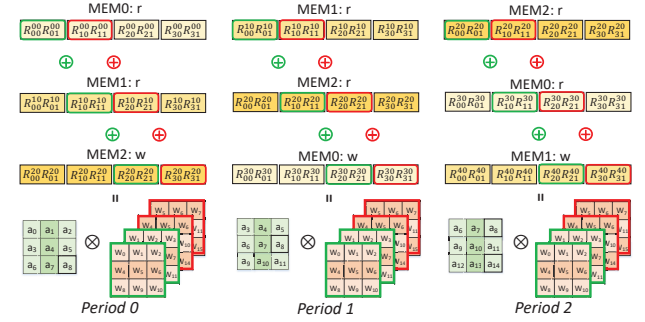


Fig. 8. ERRR used in the DCNN. r and w represent the read and write states of the memory. The PSums in the green and red rectangles are added to produce the results for transferred filters in the upper and lower meta filters, respectively. \otimes means the CONV process.

In the SCNN, two PSums are produced in one transferred filter row, as shown in Fig. 7. Therefore, the 3×3 transferred filter generates 6 PSums each time, every two of which are stored in one memory bank. Likewise, the results of different input rows are stored in different memory locations. In period 0, the PSums of the first input row are stored in MEM0, the second row in MEM1 and the third row in MEM2, as shown in Fig. 9. The PSums in the green rectangles are added to produce the results for one original transferred filter and its horizontal-symmetric version. Moreover, there is a transferred filter that is the vertical-symmetric version of the original transferred filter (in Fig. 4(b)). The PSums in red rectangles can be added to form results for this transferred filter and its horizontal-symmetric version. In the subsequent periods, the memory and PSums will be cyclically reused, similar to ERRR in the DCNN, which can eliminate potential computations and improve the CONV processing speed.

IV. TFE HARDWARE ARCHITECTURE

To reap the benefits of the two proposed techniques, an energy-efficient TFE hardware architecture is proposed for the compressed CNN acceleration. The overall TFE hardware architecture is illustrated in Fig. 10. The architecture consists of a memory system, a 16×16 PE array, an SR group and a top control unit. The filters and images are loaded from the off-chip memory to the on-chip memory for 16-bit CONV layers and FC layers on the PE array. After computations, the products from the PE array are written into the SR group to obtain PSums of the CONV process in each row.

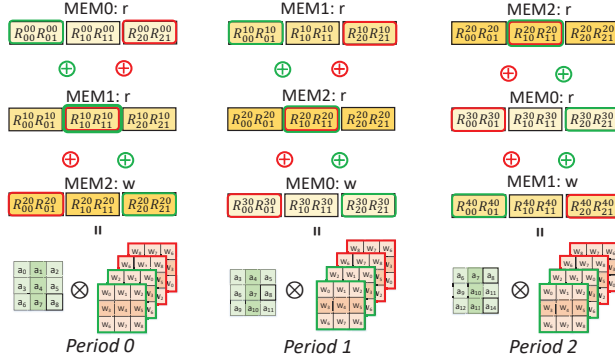


Fig. 9. ERRR used in the SCNN. The PSums in the green and red rectangles are added to produce the results for the two transferred filters and their horizontal- and vertical-symmetric versions.

Next, these PSums are written into the memory system to generate results in CONV windows, which are finally stored in off-chip memory. This computation flow is scheduled by the top control unit. The FC layers, which accomplish the mapping from high-level features to the final results, are also supported by the proposed hardware to achieve end-to-end CNN implementations. Inspired by previous methods [35], [56], we implement FC layers in a CONV manner. Note that FC layers consume only approximately 1.00% of the computations of the overall system on average [2], [35], [54]. Therefore, the same hardware unit is assigned for CONV and FC layers, which has almost no impact on the total runtime.

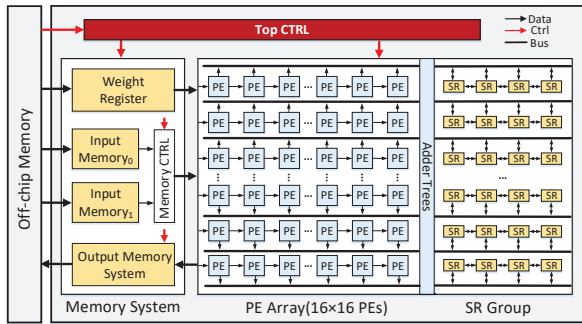


Fig. 10. TFE hardware architecture.

Sub-array-based filter mapping method (SAFM). Currently, most solutions take a single PE as the basic unit to fulfill various filters in a systolic manner. Though high flexibility is achieved, complex control logic and interconnection overhead are incurred. In the TFE hardware architecture, we propose SAFM, which uses static 3×3 and 4×4 PE sub-arrays as the basic unit to support various filter mappings, as shown in Fig. 11. For example, 5×5 and 6×6 filters can be substituted by four 3×3 PE sub-arrays. In addition, 7×7 filters can be mapped into four 4×4 PE sub-arrays. For 11×11 filters in the first layer of AlexNet, they can be first partitioned into nine 4×4 small filters as proposed in [44], which can be supported by nine 4×4 PE sub-arrays. Therefore, the complexity of

the control logic and data communication are substantially reduced with marginal overhead on flexibility and hardware utilization. Moreover, the systolic PE array is always used in conventional hardware architectures, where input/output data and weights are transferred PE by PE at each cycle. Namely, the same group of input data and weights used in the current PE will also be required by its neighbor PE at the next cycle. Therefore, sufficient registers are required to store these data (sometimes the entire-row input/output data in Eyeriss), and non-negligible control logic is needed for data communication among PEs. In SAFM, input data is broadcasted in each PE sub-array and output data will be simultaneously generated and summed up outside the PE array, which can reduce output register numbers and accesses. Furthermore, the complexity of the control logic for data broadcast in the PE sub-array is much lower than that of data communication among single PEs. To decrease fan-out and maintain accurate sequential logic, each PE sub-array is connected to one independent broadcast register and input data is read from this register to the PE sub-array. As 3×3 and 4×4 PE sub-arrays are used as the basic unit, only two groups of broadcast registers are assigned.

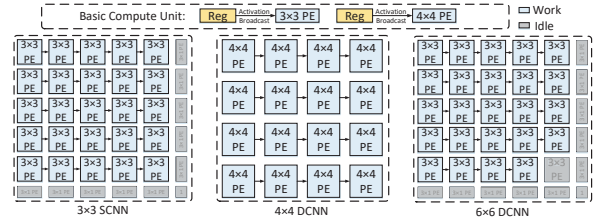


Fig. 11. 3×3 and 4×4 PE sub-arrays and their support for various filter.

Processing Element (PE). The microarchitecture of one PE in the TFE hardware architecture is shown in Fig. 12. Each PE is equipped with a weight register for a single weight, as each weight should be kept for a while in the proposed dataflow, an input register for a single broadcast input data and a PSum register for output data in the conventional CONV process. The clock gating technique is used to avoid multipliers' unnecessary toggling when the weight or input is zero, which can further save approximately 5% of dynamic power. This limited dynamic power saving is mainly because of the reduction of zero weight values in the transferred filters, and the PE array consumes a relatively low fraction of overall power. Additionally, there is an adder, a multiplier and a multiplexer in the PE. Each PE can support both the transferred-filter and conventional CONV processes. When the implementation mode is the conventional CONV process, the inner product will be sent to the adder and added with the data from the multiplexer. The data is the PSum or product from the left-neighbor PE when the current PE is on, and the corresponding sum is stored in the PSum register ready for the right-neighbor PE at the next cycle. In contrast, the data is zero when the current PE is off. When the implementation mode is the transferred-filter CONV process, the product from the multiplier will be directly sent to the adder trees through the

data bus. The products that correspond to the same ofmap will be summed up by the adder trees. The sums will be transferred to the SR group to implement PPSR.

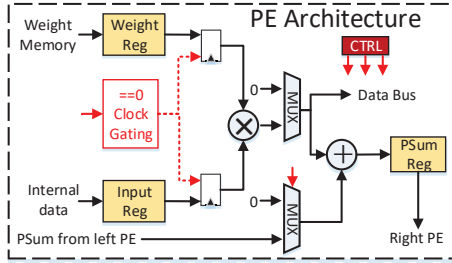


Fig. 12. Microarchitecture of the PE module.

Stacked Register (SR) Group. Each SR consists of several registers and is used to implement PPSR, as shown in Fig. 5 and 6. Additionally, there are 16×16 PEs in the PE array, which can be partitioned into several PE sub-arrays to support multiple ifmaps and ofmaps in parallel. Referring to the conventional CNN dataflow, each ifmap requires independent SRs, which means that we should assign one SR to each PE. However, a higher power and a larger area will be consumed due to the large demand arising from register numbers and accesses. As one ofmap requires several ifmaps, we propose to pre-add the PSums of different ifmaps that correspond to the same ofmap. Then, their sums are transferred into the SR group to implement PPSR. In this way, the required number of SRs depends on the number of ofmaps instead of ifmaps, which can reduce the SR consumption and register access by 85.9%. Based on the actual PE array partition, 6×6 SRs are assigned to the SR group to support different transferred filters.

Input Memory/Weight Register. A 512 B weight register is used to store the required filters from off-chip memory. For different CONV processes, the corresponding weights are read from the weight register and assigned into the PE array. To save output memory size and increase data re-usage, once the calculation of each row of the ofmaps is finished, the same row in another group of ofmaps will be computed. Only when the current row in all the ofmaps is obtained will another row of computations in the ofmaps start. As the number of PEs are limited, input data and weights will be partly read in. Therefore, the weights are kept unchanged until the corresponding input data for one row of one ofmap has been processed. As there is enough time to load another 256 weights from the off-chip memory, only one of the weight registers is needed in our architecture. In contrast, a ping-pong input memory (4 KB each) is set to pre-load input data from off-chip memory and read data into the PE array simultaneously. The reason is that as the input data should be updated every cycle, data from off-chip memory is written in one piece of input memory and the data in the other piece is loaded to the PE array. When the data in one piece of input memory is used up by the PE array, it will be used to store data from the off-chip memory, and the other piece is used to provide for the

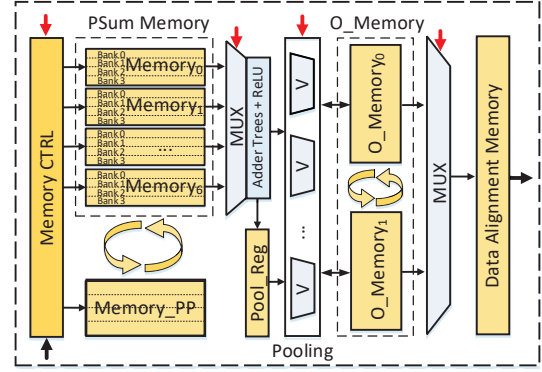


Fig. 13. Architecture of the output memory system.

PE array, namely, the two pieces of input memory work in ping-pong mode.

Output Memory System. The output memory system implements the ERRR technique and generates final activations of ofmaps. The hardware architecture of ERRR is shown in Fig. 13. First, the PSums from the PE array are stored in PSum memories, as shown in Fig. 8 and 9. There are seven 8 KB PSum memories (formed of four 2 KB banks) in total to support the maximum 7×7 filter. As multiple ifmaps are required to produce one ofmap and the PE array can only process part of the ifmaps, an 8 KB ping-pong memory, called Memory_PP, is used to alternately read and write intermediate PSums in PSum memories. The number of required PSum memories is decided by the filter size. The PSums in these PSum memories are read, added to adder trees and activated by the ReLU function, which then outputs activations of ofmaps.

Usually, the CONV layer is followed by one pooling layer. As output activations of ofmaps are output row by row in the proposed architecture, we first pool output activations row by row. The activations that will be pooled in one row are first stored in a register, named Pool_Reg. For example, in one 2×2 pooling layer, one activation is stored in Pool_Reg. When the neighbor activation is output, the activation in Pool_Reg is read out, and both are transferred to pooling units to perform a pooling operation. The pooling result is written into output memory, named O_Memory in Fig. 13. After output activations of the next row is produced and executed with a 1×2 pooling operation, the results are sent to the pooling unit along with activations read out from the O_Memory to complete a 2×2 pooling operation. It can be learned that there are both reading and writing operations in the pooling operation simultaneously, so two 1.0 KB memories (O_Memory₀ and O_Memory₁) are used in O_Memory to support the pooling operation.

When the pooled activations in one output memory are full, a multiplexer (MUX) is used to choose this memory and transfer its data into a 16 KB Data Alignment Memory (DAM). If the channel number of ofmaps in the DAM does not equal to the channel number of ifmaps that required in the subsequent CONV layer implementation at one time, then these output activations are temporarily stored in the DAM.

Otherwise, they are written back into off-chip memory. In this way, the complex data alignment operation is eliminated, which can reduce the control complexity and memory access.

V. EXPERIMENTAL RESULTS

In this section, the evaluation methodology is first introduced. Second, technical specification comparisons and analysis of the TFE and the Eyeriss hardware architecture are provided. Third, the performance is compared with that of Eyeriss and other recent related works. Fourth, TFE is compared with Eyeriss, UCNN [19] and SnaPEA [2] in terms of energy efficiency (performance/energy consumption) to further demonstrate the proposed architecture’s advantages. Finally, the effects achieved by implementing PPSR and ERRR are evaluated, along with off-chip memory access savings.

A. Methodology

To prove the generality of the TFE, some popular and state-of-the-art networks [17], [26], [42], [47] are used in this work. The networks are first converted to the transferred filter-based networks and pre-trained in Python on the TensorFlow platform. The evaluation dataset is ImageNet [12], which is public and widely used in image classification tasks. The top-1 accuracy of the four transferred networks is listed in Table V, and all accuracy losses can be maintained within 1%.

TABLE II
TOP-1 ACCURACY OF ORIGINAL AND TRANSFERRED NETWORKS ON
IMAGENET

	AlexNet	VGGNet	GoogLeNet	ResNet
Original	53.60%	70.94%	68.21%	76.92%
DCNN _{4×4}	53.24%	70.25%	67.75%	76.11%
SCNN	53.46%	70.54%	67.92%	76.34%

As the Eyeriss [6] architecture is widely used as the baseline in many works [2], [15], [53] and only relative results are provided in these works, we also choose it as the baseline for a fair comparison. Furthermore, the same data width format (16 bit), clock frequency (200 MHz, maximum frequency is 288 MHz) and technology (TSMC 65 nm) that used in the [6] are used in this work. For speed comparison, the computational unit numbers are normalized to be the same in all compared architectures with hardware utilization taken into consideration. For energy efficiency comparison, unlike other works that estimate the power consumption of Eyeriss [2], [14], [15], [53], the power consumptions on VGGNet and AlexNet are directly extracted from the Eyeriss paper [6] and used here to compare the energy efficiency, which can mitigate the disparity of possible measurement errors.

We use the Synopsys Design Compiler and a TSMC 65 nm standard-cell library to synthesize the proposed architecture and obtain the power consumption, area and performance after place-and-routing. Referring to [2], the publicly available Micron’s DDR4 system power calculator [1] is used to estimate the power cost of access to off-chip memory.

TABLE III
TECHNICAL SPECIFICATIONS OF THE TFE AND EYERISS HARDWARE
ARCHITECTURE.

Architecture	TFE	Eyeriss*
Technology	TSMC 65nm 1P8M	TSMC 65nm 1P9M
Voltage	1 V	1 V
Frequency	200 MHz	200 MHz
Memory	160.0 KB	181.5 KB
#PEs	256	168
Area	7.1 mm ²	12.25 mm ²
Power	62 mW	257 mW

*The features are extracted from the paper [6].

B. Hardware Verification

Based on the TSMC 65 nm technology, the detailed technical features of the TFE hardware architecture are listed in Table III. Note that the power consumptions are the averaged results for VGGNet and AlexNet. Although a similar memory size and more PEs are used, the TFE hardware architecture can save more area (1.73×) and power (4.15×) consumption than Eyeriss. As analyzed in Section IV, this is mainly attributed to four reasons.

First, in Eyeriss, the entire input row is processed in one PE and the Psums of each row are obtained inside the PE array, which requires registers for these input data and the entire output results. In contrast, SAFM used in the TFE hardware architecture adopts 3×3 and 4×4 PE sub-arrays, instead of a single PE, as the basic unit to support various filters. One input data for each PE sub-array is broadcast to each PE and the products are simultaneously generated, which only consumes two registers for a pair of input data and weight in each PE. Second, as PPSR and ERRR facilitate communicating data in a very regular and systolic manner, the complexity of the data communication and control logic can be efficiently decreased. Furthermore, the ofmaps are figured out row by row in the TFE, without a requirement for complex data alignment as in Eyeriss. Third, products from each PE are summed inside the PE array, and the Psums are first stored in the local Psum register in Eyeriss. In contrast, the Psums from the PE sub-array that correspond to the same ofmaps will be added up and then written into the SRs. The SRs are stripped from each PE and form an SR group outside the PE array, which can further reduce the SRs and register access by 85.9%. These designs can significantly decrease the area and power consumption. Fourth, partial sums read from the register or memory can be reused for different filters in PPSR and ERRR, and therefore, a high data reuse rate is achieved, significantly decreasing memory access and computations.

The main overhead of the TFE hardware architecture comes from the memory and register access, which consumes a total of 69.3% of the area and 75.0% of the power, as shown in Fig. 14. In contrast, the PE array consumes approximately 16.5% of the area and 21.1% of the power. The reason is that amount of redundant computations are eliminated and replaced by the memory and register access. Benefiting from the high data reuse in PPSR and ERRR, the absolute energy consumption of memory and register access is far less than that

in conventional computations. Another overhead relates to the specific control logic and data communication. Owing to the the proposed regular dataflow and SAFM, the corresponding power consumption is constrained to lower than 1.2%, with an 8.8% area consumption.

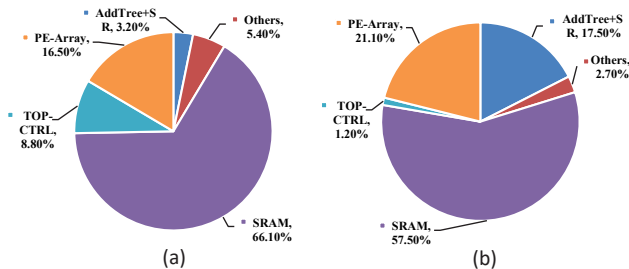


Fig. 14. (a) Area breakdown and (b) power breakdown of the TFE.

C. Performance Improvement

Eyeriss and other recent related works, including weight-compression-based methods (weight clustering [16], weight pruning [37], [51] and repetitive weight elimination [19]) and computation-reduction-based methods (prediction [2], Winograd [53] and asymmetric convolution [4]), are used for comparison with TFE. All of these works can be used to improve the performance of the CNN implementation.

1) *Comparison with Eyeriss*: The hardware speedup values of CONV layers in different networks over Eyeriss are shown in Fig. 15(a). As 3×3 filters are used in VGGNet and ResNet, each CONV layer can benefit from the TFE, which can speed up their CONV process by $2.2\times$ and $2.12\times$ in the 4×4 DCNN and $3.45\times$ and $3.39\times$ in the SCNN. As the 11×11 filter in the first layer of AlexNet has a great effect on the final accuracy, the filter is maintained unchanged in the DCNN, which degrades the CONV speedup. As there are many 1×1 filters in GoogLeNet, which cannot be transferred, the speedup in TFE is also limited compared to other state-of-the-art networks. Additionally, as both 3×3 and 5×5 filters are widely used in GoogLeNet, heterogeneous meta filters are adopted to maintain acceptable accuracy in the DCNN implementation, which causes less speedup than that in SCNN. Finally, $2.07\times$ (4×4 DCNN), $2.93\times$ (6×6 DCNN) and $3.17\times$ (SCNN) speedup values, on average, are achieved, which are much better than other architectures [2], [16], [51], [53].

Fig. 15(b) illustrates the overall speedup of the TFE across different networks including both CONV and FC layers. As the transferred filter-based methods have no effect on FC layers, the overall speedup is degraded in all networks. However, FC layers account for a very small proportion of total computations in all networks except for AlexNet. Therefore, the loss in speedup is very limited, less than 3%. For AlexNet, where FC layers consume more than 8% of the computations, the loss in speedup is greater than 9.8% compared with the speedup of CONV layers. On average, the overall speedup values degrade by 3.9% (4×4 DCNN), 6.9% (6×6 DCNN) and 6.3% (SCNN)

compared to CONV layer speedup, which is mainly caused by the high speedup degradation in AlexNet.

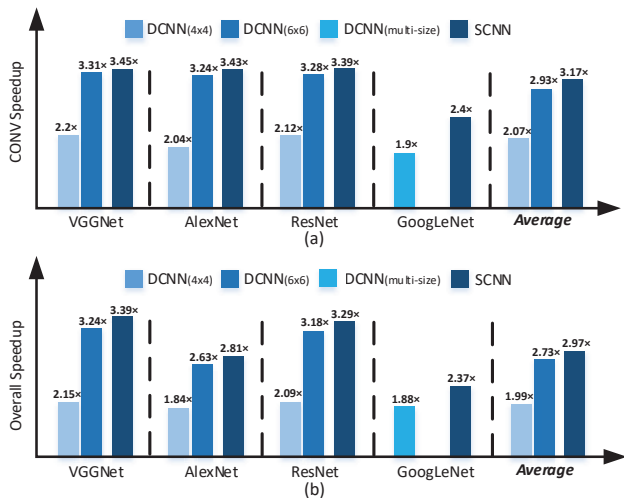


Fig. 15. (a) CONV layers and (b) overall speedup with the DCNN and SCNN based on the TFE.

2) *Comparison with weight-compression-based methods*: The performance comparisons with four recent weight-compression-based methods, Han [16], SSL [51], ADMM [37], and UCNN (50% weight sparsity) [19], are given in Fig. 16, where the baseline of all methods is Eyeriss. Without loss of generality, AlexNet, a common network used in these methods, is chosen as the evaluation network. Though a high parameter reduction ratio is achieved by the four weight-compression-based methods, their actual speedups in the hardware implementation do not match their high parameter reduction ratio. The reason is that their irregular zero value distribution in both activations and weights is seriously unfriendly for hardware implementation, including complex control logic, irregular data access, encoding-decoding operation and so on. These overheads not only significantly degrade their speedups but also cause higher power consumption. In contrast, the regular CONV process in the proposed TFE can significantly alleviate this issue. With the SCNN-based AlexNet implemented in the TFE, $5.36\times$ [16], $4.45\times$ [51] and $3.24\times$ [19] speedups are achieved. Though the speedup is marginally lower than that in [37], we strongly believe that with much more regular CONV processes in the TFE hardware architecture, lower power consumption and higher energy efficiency can be achieved compared with these weight-compression-based methods.

3) *Comparison with computation-reduction-based methods*: The comparison results with three recent computation-reduction-based methods, SnaPEA [2], the Winograd algorithm [53] and asymmetric convolution [4], are shown in Fig. 17. As VGGNet is relatively more accurate in various tasks, the evaluation is implemented on VGGNet. For other popular networks, the comparison results are similar.

Although no accuracy loss occurred, the Winograd algorithm utilizes nearly $1.7\times$ more parameters than the original

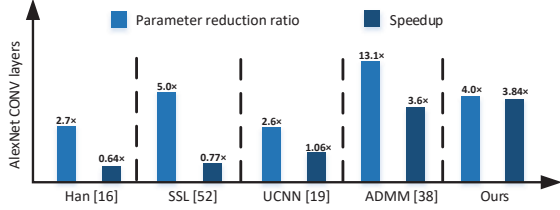


Fig. 16. Parameter reduction ratio and speedup comparison results with compression-based methods on the CONV layers of AlexNet.

networks to reduce computations of CONV layers. Furthermore, the Winograd algorithm only works well when the filter and input tile size are small (such as the 3×3 filter size and 4×4 input size), which prevents its wide usage in practice. The TFE is compared with SnaPEA under the condition of similar accuracy loss. As shown in Fig. 17, there is no parameter reduction in SnaPEA, while $2.27 \times$ (4×4 DCNN) and $4.0 \times$ (6×6 DCNN and SCNN) reductions are achieved in the TFE. Moreover, at the cost of a 1.0% accuracy loss, the TFE achieves a $2.61 \times$ speedup compared with SnaPEA. When the SCNN-based VGGNet is implemented, not only a $2.72 \times$ speedup but also a 0.6% higher accuracy are achieved compared with SnaPEA. The asymmetric convolution method [4], transferring the 3×3 filter into the 3×1 and 1×3 filters, is widely researched and used. Though a similar accuracy loss is maintained, the asymmetric method consumes at least $1.51 \times$ (4×4 DCNN) and $2.67 \times$ (SCNN) more parameters. In particular, its CONV layer speedup is significantly lower than ours, by $1.47 \times$ (4×4 DCNN) and $2.21 \times$ (SCNN). In summary, the experimental results show that compared with related computation-reduction-based methods with similar accuracy, the proposed TFE can not only compress CNN model size but also significantly reduce repetitive computations. Thus, our proposed TFE is very appropriate for accelerating CNNs for resource-constraint platforms.

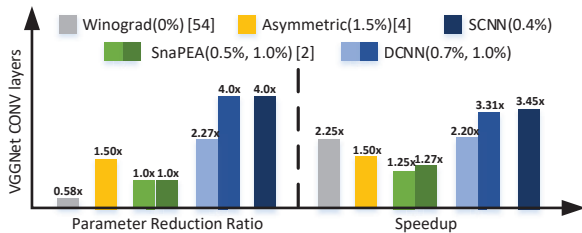


Fig. 17. Parameter reduction ratio and speedup comparison results with computation-reduction-based works on CONV layers of VGGNet. The percentages inside parentheses are the corresponding accuracy loss.

4) *Comparison with other related methods:* As ResNet and GoogLeNet are not used in all related works, the results are assembled and listed in Table IV. With the SCNN adopted in the TFE, $1.6 \times$ (SnaPEA [2]) and $1.19 \times$ (Multi-CLP [41]) overall speedup improvements are obtained on GoogleNet; $2.19 \times$ (UCNN [19]) and nearly the same (Bitfusion [40]) speedup are achieved on ResNet.

Especially, compared with SCNN-Nvidia [36], which takes sparsity advantage of both weight and activations, $1.14 \times$ (GoogLeNet), $1.56 \times$ (AlexNet) and $1.05 \times$ (VGGNet) speedups in CONV layers are achieved. Note that the networks used in SCNN-Nvidia are pre-pruned for high sparsity, while non-pruned networks are adopted in the TFE.

TABLE IV
OVERALL SPEEDUP OF OTHER RELATED METHODS OVER EYERISS ON GOOGLNET AND RESNET

	UCNN [19]	Bitfusion [40]	TFE _{SCNN}
ResNet	1.50x	3.62x	3.29x
	SnaPEA [2]	Multi-CLP [41]	TFE _{SCNN}
GoogLeNet	1.48x	2.00x	2.37x

To further prove the effect of the proposed TFE, three more recent networks, DenseNet [21], SqueezeNet [22] and Residual Attention Network (ResANet) [50], are evaluated with different transferred-filter-based algorithms. The corresponding accuracy losses are all less than 1%. Note that as almost no related works implement these recent networks, only our speedups of CONV layers and overall network are listed in Table V. Likewise, the TFE can achieve considerable speedup of CONV layers on SqueezeNet and ResANet, $2.09 \times$ and $2.22 \times$ on average, respectively. In contrast, On DenseNet, as the 1×1 filter-related computations constitute approximately 60% of the total computations, only an average speedup of $1.35 \times$ is achieved. Because the FC layer accounts for very small proportion of overall computations, marginal degradation is caused for overall speedup in all evaluated networks. Actually, most recent networks have similar CONV operations to the networks evaluated in this work [25], excluding depth-wise convolution-based networks such as MobileNet, which further demonstrates that the proposed TFE can accelerate most mainstream and recent networks.

TABLE V
CONV LAYER/OVERALL SPEEDUP OF THE TFE ON DENSENET, SQUEEZENET AND RESANET

	DenseNet [21]	SqueezeNet [22]	ResANet [50]
DCNN $_{4 \times 4}$	1.29x/1.24x	1.65x/1.62x	1.48x/1.39x
DCNN $_{6 \times 6}$	1.38x/1.31x	2.30x/2.26x	2.54x/2.44x
SCNN	1.39x/1.32x	2.32x/2.30x	2.64x/2.55x

D. Energy Efficiency Improvement

To further prove the efficiency of the TFE, the energy efficiency (performance/energy consumption) is compared. Because the energy efficiency is not given in most related methods, a weight-compression-based method, UCNN (50% weight sparsity) [19], and a computation-reduction-based method, SnaPEA [2], are used for comparison. Likewise, Eyeriss is chosen as the baseline of all methods. As only VGGNet and AlexNet are evaluated for Eyeriss, these two networks are used here for comparison.

Fig. 18 depicts the overall energy efficiency improvement of the TFE, UCNN and SnaPEA (predictive mode) over Eyeriss.

The accuracy loss (compared with the original accuracy) in all methods is maintained within 1%. As the proposed PPSR and ERRR achieve high performance (Sections 3 and 5.3) and the TFE hardware architecture consumes much less power (Sections 4 and 5.2), both improvements are merged in the TFE to achieve higher energy efficiency on VGGNet and AlexNet than these related works. On average, the energy efficiency of the SnaPEA architecture is $1.48\times$ higher than that of Eyeriss, while that of UNCC is $4.23\times$ higher. Compared with SnaPEA, our architecture can further improve the energy efficiency by $5.62\times$ (4×4 DCNN), $8.55\times$ (6×6 DCNN) and $8.99\times$ (SCNN). Likewise, the energy efficiency is improved by $1.71\times$ (4×4 DCNN), $2.53\times$ (6×6 DCNN) and $2.69\times$ (SCNN) on AlexNet compared with UCNN [19] (VGGNet is not evaluated for UCNN). As more computations and parameters are saved by the SCNN and 6×6 DCNN, their energy efficiencies are higher than that of the 4×4 DCNN. Additionally, the SCNN achieves $1.05\times$ higher energy efficiency than the 6×6 DCNN, though with the same reduction of computations and parameters. The reason is that the SCNN involves less register access than the 6×6 DCNN, as shown in Fig. 5 and 6. Additionally, the hardware utilization of the SCNN is higher than that of the 6×6 DCNN, yielding a higher speedup.

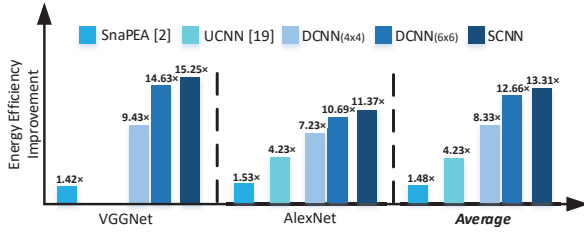


Fig. 18. Overall energy efficiency improvement by the TFE compared with Eyeriss and SnaPEA.

E. Ablation Analysis

First, the MAC reductions of PPSR and ERRR are verified on VGGNet, and the corresponding results are shown in Fig. 19. As the width and height of meta filters in the DCNN are always equal, the same benefits can be obtained in PPSR and ERRR. Namely, the same MAC reductions, $1.5\times$ (4×4 DCNN) and $2.0\times$ (6×6 DCNN), are achieved by these two proposed techniques. In the SCNN, PPSR can reduce competitive MACs among horizontal-symmetric transferred filters, and ERRR can be of benefit to vertical-symmetric filters. However, either technique can only accelerate two of eight filters, as the 180° and 270° rotated filters require both horizontal- and vertical-symmetric operations. Therefore, only the combination of PPSR and ERRR can achieve the maximal MAC reduction of $4.0\times$.

Furthermore, as the TFE can reduce redundant weights, less weights are loaded than that in the conventional architectures without model compression, decreasing the overhead caused by power-intensive off-chip memory access. As shown in Fig. 20, on VGGNet, AlexNet and ResNet, the off-chip memory access can be reduced by $1.28\times \sim 1.38\times$ (4×4 DCNN),

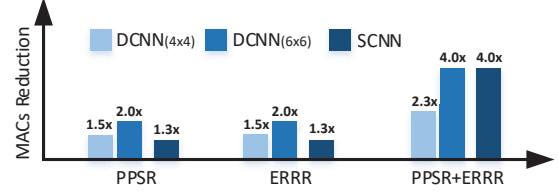


Fig. 19. Computations of the DCNN and SCNN with/without PPSR and ERRR.

$1.48\times \sim 1.59\times$ (6×6 DCNN) and $1.48\times \sim 1.60\times$ (SCNN). As there are many 1×1 filters in GoogLeNet that cannot be transferred, the corresponding off-chip memory access cannot be saved. However, the TFE still reduces the off-memory access by $1.19\times \sim 1.24\times$ compared with the conventional architectures, which demonstrates the effectiveness of the TFE in the off-chip memory access reduction.

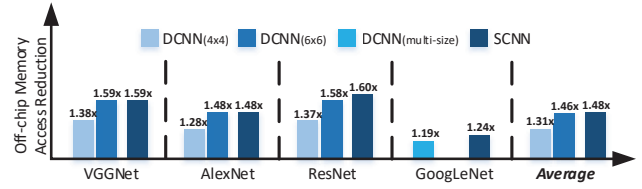


Fig. 20. Off-chip memory access required by the transferred filters compared with the original filters.

Factor effectiveness analysis. CReLU, MBA and SCNN implemented in TFE have the same compression and acceleration ratio on canonical CONV layers with different filter sizes. In contrast, when the meta filter size in DCNN is fixed, the compression and acceleration ratio decreases along with the increasing of the original filter size. To quantitatively measure the corresponding compression and acceleration ratio, the parameter and MAC number of networks are formulated referring to Table I.

The number of parameters (NUM_P_O) and MAC operations (NUM_M_O) in an original CNN layer is defined as

$$\begin{aligned} \text{NUM_P_O} &= N \times M \times K \times K, \\ \text{NUM_M_O} &= E \times F \times N \times M \times K \times K. \end{aligned} \quad (1)$$

while the direct implementation of DCNN consumes parameter number (NUM_P_D) and MAC number (NUM_M_D):

$$\begin{aligned} \text{NUM_P_D} &= \frac{M}{(Z-K+1)^2} \times N \times Z \times Z, \\ \text{NUM_M_D} &= \frac{E \times F \times M}{(Z-K+1)^2} \times N \times (Z-K+1)^2 \times K \times K \\ &= E \times F \times N \times M \times K \times K. \end{aligned} \quad (2)$$

In the TFE, the parameter number (NUM_P_T) and MAC number (NUM_M_T) are

$$\begin{aligned} \text{NUM_P_T} &= \frac{M}{(Z-K+1)^2} \times N \times Z \times Z, \\ \text{NUM_M_T} &= \frac{E \times F \times M}{(Z-K+1)^2} \times N \times (Z)^2 \\ &= \frac{E \times F \times M \times (Z)^2 \times N}{(Z-K+1)^2}. \end{aligned} \quad (3)$$

It can be learned that NUM_P_T is equal to NUM_P_D but less than NUM_P_O. The reduction proportion is as follows

$$\frac{NUM_P_O}{NUM_P_T} = \frac{(Z - K + 1)^2 \times (K)^2}{(Z)^2}. \quad (4)$$

In contrast, NUM_M_T is reduced significantly compared with NUM_M_O and NUM_M_D as follows

$$\frac{NUM_M_O}{NUM_M_T} = \frac{NUM_M_D}{NUM_M_T} = \frac{(Z - K + 1)^2 \times (K)^2}{(Z)^2}. \quad (5)$$

According to Eq. 4 and Eq. 5, the parameter and MAC reduction proportions mainly depend on meta filter size Z and original filter size K . To improve the parameter and MAC reduction benefits, namely higher compression and acceleration ratio, Z should be as large as possible when K is fixed. However, larger Z may cause higher accuracy loss. On the other hand, when meta filter size Z is fixed, and when original filter size $K = \frac{Z+1}{2}$, both largest parameter reduction ratio and MAC reduction ratio can be obtained by TFE according to Eq. 4 and 5. As Z cannot be too large for acceptable accuracy loss (when Z is set as 6, accuracy loss approaches 2% in all evaluated networks), K is often larger than $\frac{Z+1}{2}$. Therefore, the parameter and MAC reduction ratio, namely compression and acceleration ratio, often decrease with K increased.

Because SCNN and DCNN in TFE can only implement 1×1 , depth-wise CONV layer and FC layer directly without compression and acceleration, networks containing more such layers benefit less benefits from TFE. In the worst case, if all layers of a network are 1×1 CONV layers, TFE just has the same performance as the conventional processors. Namely, as long as canonical CONV layers with larger filters exist in networks, TFE can compress and accelerate these networks and outperform conventional processors. Actually, in most mainstream networks, canonical CONV layers with larger filters consume relatively more MACs to keep high accuracy. Therefore, these mainstream networks are good target for TFE to be compressed and accelerated. Moreover, as TFE focuses on each layer independently, the overall compression and acceleration ratio are fixed when the ratio of canonical CONV layers to other layers (e.g. 1×1 CONV layers) is unchanged (assuming the proportion of canonical CONV layers with different filter size are unchanged). Consequently, the overall compression and acceleration ratio are non-correlated with the feature map size and overall number of layers.

VI. RELATED WORK

CNN acceleration. Most CNN architectures [3], [23], [28], [39], [54] propose customized processing flows to maximize data reuse and minimize memory access, especially off-chip memory access. However, the MAC operations in these CNN architectures are maintained the same as in the original CNNs. Recently, some architectures utilize the property of the ReLU function to decrease computations that will have negative results [2], [45]. However, the algorithm accuracy will be much lower with an increased reduction of computations. The Winograd algorithm optimizes the CONV flow to explore the redundant computation with no accuracy loss [32], [53].

However, this method is only applicable to computing the CONV of small filters for small input sizes. The asymmetric convolution method [4] decomposes 3×3 CONV operation into 1×3 and 3×1 CONV operations and has achieved significant acceleration on object recognition. However, this method imposes strict constraints on the training process, ignoring correlations among filters, degrading the final accuracy.

CNN compression. The weight-compression-based methods usually adopt weight quantization and pruning in one simple (re-)training procedure [10], [16], [18], [48], [51], thereby achieving effective model compression. A classic weight-pruning work is [16]. It uses a heuristic, iterative method to prune the weights with small magnitudes. The state-of-the-art weight quantization technique adopts an iterative quantization and retraining framework, with randomness incorporated into the quantization [10]. However, weight-compression-based methods require manually determining parameter settings and encounter difficulty in converging. Additionally, the pruned CONV process is irregular; namely, at least one index per weight is needed to index the relative location of the next weight. Therefore, the corresponding control complexity is much higher than that in other methods [11], [27], [37].

VII. CONCLUSION

This work proposes the TFE to compress and accelerate CNN models. The TFE employs two hardware-friendly techniques, PPSR and ERRR, to reduce all repetitive computations in transferred filter-based algorithms, which can significantly benefit CNN model acceleration. Moreover, the model compression property of the TFE can reduce memory size requirements and memory access. Furthermore, the TFE hardware architecture is developed to support various CNN models with less area and power consumption. The evaluation results show that compared with the Eyeriss architecture, our proposed architecture achieves average $1.99 \times$ (4×4 DCNN), $2.73 \times$ (6×6 DCNN) and $2.97 \times$ (SCNN) overall speedups on most state-of-the-art networks, significantly outperforming other related works with a similar accuracy loss (less than 1%). Furthermore, average energy efficiency improvements of $8.33 \times$ (4×4 DCNN), $12.66 \times$ (6×6 DCNN) and $13.31 \times$ (SCNN) are achieved on VGGNet and AlexNet compared with Eyeriss, and the results are $5.63 \times$, $8.55 \times$ and $8.99 \times$ higher, respectively, than those achieved with the SnaPEA architecture.

ACKNOWLEDGMENT

This work is supported in part by the National Natural Science Foundation of China (Grant No.61834002), and in part by the National Key R and D Program of China (Grant No. 2018YFB2202101), and in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China (Grant No. 2018ZX01028201).

REFERENCES

- [1] "DDR4 Spec - Micron Technology, Inc," <http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/>.

- [2] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "Snapec: Predictive early activation for reducing computation in deep convolutional neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 662–673.
- [3] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda *et al.*, "Brein memory: A 13-layer 4.2 k neuron/0.8 m synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos," in *2017 Symposium on VLSI Circuits*. IEEE, 2017, pp. C24–C25.
- [4] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H.-J. Yoo, "14.6 a 0.62 mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 248–249.
- [5] R. Chen, Y. Chen, and J. Su, "Deep convolutional neural networks compression method based on linear representation of kernels," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041. International Society for Optics and Photonics, 2019, p. 110412N.
- [6] 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, Conference Proceedings, pp. 367–379.
- [7] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.
- [8] —, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.
- [9] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*, 2016, pp. 2990–2999.
- [10] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.
- [11] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 749–763.
- [12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [13] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, "Exploiting cyclic symmetry in convolutional neural networks," *arXiv preprint arXiv:1602.02660*, 2016.
- [14] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 751–764.
- [15] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 807–820.
- [16] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
- [17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [18] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800.
- [19] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 674–687.
- [20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.
- [21] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.
- [22] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size," *ArXiv*, vol. abs/1602.07360, 2017.
- [23] Z. Jiang, S. Yin, M. Seok, and J.-s. Seo, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," in *2018 IEEE Symposium on VLSI Technology*. IEEE, 2018, pp. 173–174.
- [24] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.
- [25] A. Khan, A. Sohail, U. Zahoor, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, pp. 1 – 62, 2020.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012.
- [27] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 821–834.
- [28] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid-State Circuits Conference-ISSCC*. IEEE, 2018, pp. 218–220.
- [29] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [30] H. Li, W. Ouyang, and X. Wang, "Multi-bias non-linear activation in deep neural networks," in *International conference on machine learning*, 2016, pp. 221–229.
- [31] S. Liu, Z. Li, T. Li, V. Srikumar, V. Pascucci, and P. Bremer, "Nlize: A perturbation-driven visual interrogation tool for analyzing and interpreting natural language inference models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 651–660, Jan 2019.
- [32] L. Lu and Y. Liang, "Spwa: an efficient sparse winograd convolutional neural networks accelerator on fpgas," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.
- [33] X. Miao, X. Zhen, X. Liu, C. Deng, V. Athitsos, and H. Huang, "Direct shape regression networks for end-to-end face alignment," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5040–5049.
- [34] C. Min, A. Wang, Y. Chen, W. Xu, and X. Chen, "2pfpce: Two-phase filter pruning based on conditional entropy," *arXiv preprint arXiv:1809.02220*, 2018.
- [35] H. Mo, L. Liu, W. Zhu, Q. Li, H. Liu, W. Hu, Y. Wang, and S. Wei, "A 1.17 tops/w, 150fps accelerator for multi-face detection and alignment," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 80.
- [36] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–40, 2017.
- [37] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 925–938.
- [38] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *international conference on machine learning*, 2016, pp. 2217–2225.
- [39] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," in *Proceedings of the 45th*

- Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 764–775.
- [40] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, 2018.
- [41] Y. Shen, M. Ferdman, and P. A. Milder, “Maximizing cnn accelerator efficiency through resource partitioning,” *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 535–547, 2017.
- [42] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Computer Science*, 2014.
- [43] S. Son, S. Nah, and K. Mu Lee, “Clustering convolutional kernels to compress deep neural networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 216–232.
- [44] L. Song, W. Ying, Y. Han, and X. Li, “C-brain: a deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization,” in *Design Automation Conference*, 2016.
- [45] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, “Prediction based execution on deep neural networks,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 752–763.
- [46] S. Sun, Y. Cheng, Z. Gan, and J. Liu, “Patient knowledge distillation for bert model compression,” *arXiv preprint arXiv:1908.09355*, 2019.
- [47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [48] K. Ullrich, E. Meeds, and M. Welling, “Soft weight-sharing for neural network compression,” *arXiv preprint arXiv:1702.04008*, 2017.
- [49] J. Vongkulbhisal, F. De la Torre, and J. P. Costeira, “Discriminative optimization: Theory and applications to computer vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 4, pp. 829–843, April 2019.
- [50] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, “Residual attention network for image classification,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6450–6458, 2017.
- [51] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
- [52] X. Xu, J. Deng, E. Coutinho, C. Wu, L. Zhao, and B. W. Schuller, “Connecting subspace learning and extreme learning machine in speech emotion recognition,” *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 795–808, March 2019.
- [53] A. Xygkis, L. Papadopoulos, D. Moloney, D. Soudris, and S. Yous, “Efficient winograd-based convolution kernel implementation on edge devices,” in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 136.
- [54] S. Yin, P. Ouyang, J. Yang, T. Lu, X. Li, L. Liu, and S. Wei, “An energy-efficient reconfigurable processor for binary-and ternary-weight neural networks with flexible data bit width,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1120–1136, 2019.
- [55] S. Zhai, Y. Cheng, Z. M. Zhang, and W. Lu, “Doubly convolutional neural networks,” in *Advances in neural information processing systems*, 2016, pp. 1082–1090.
- [56] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, “Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
- [57] H. Zhang, X. Wang, J. Zhu, and C.-C. J. Kuo, “Fast face detection on mobile devices by leveraging global and local facial characteristics,” *Signal Processing: Image Communication*, vol. 78, pp. 1–8, 2019.