Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A Survey on Acceleration of Deep Convolutional Neural Networks

Jian Cheng, Peisong Wang, Gang Li, Qinghao Hu, Hanqing Lu[1]

([1]*National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China*)

[†]E-mail: jcheng@nlpr.ia.ac.cn

Abstract: Deep Neural Networks have achieved remarkable progress during the past few years and are currently the fundamental tools of many intelligent systems. At the same time, the computational complexity and resource consumption of these networks are also continuously increasing. This will pose a significant challenge to the deployment of such networks, especially for real-time applications or on resource-limited devices. Thus, network acceleration have become a hot topic within the deep learning community. As for hardware implementation of deep neural networks, a batch of accelerators based on FPGA/ASIC have been proposed these years. In this paper, we provide a comprehensive survey about the recent advances on network acceleration, compression and accelerator design from both algorithm and hardware side. Specifically, we provide thorough analysis for each of the following topics: network pruning, low-rank approximation, network quantization, teacher-student networks, compact network design and hardware accelerator. Finally, we make a discussion and introduce a few possible future directions.

Key words: Deep Neural Networks, Acceleration, Compression, Hardware Accelerator
CLC number: TP391.4

1 Introduction

In recent years, Deep Neural Networks (DNNs) have achieved remarkable performance for a wide range of applications, including but not limited to computer vision, natural language processing, speech recognition, etc. These breakthroughs are closely related to the increased training data and more powerful computing resources. For example, one breakthrough within natural image recognition field is achieved by AlexNet (Krizhevsky *et al.*, 2012), which is trained using multiple graphics processing units (GPUs) on about 1.2M images. Since then, the performance of DNNs has been continuously improving. For many tasks, DNNs are reported to be able to outperform humans. The problem, however, is that the computational complexity as well as the storage requirements of these DNNs are also increasing drastically as shown in Table 1. Specifically, the widely used VGG-16 model (Simonyan and Zisserman, 2014) involves more than 500MB storage and over 15B FLOPs to classify a single 224×224 image.

Thanks to recent powerful GPUs and CPU clusters equipped with more memory resources and computational units, these more powerful DNNs can be trained within relatively affordable time. However, at inference time, such long executing time is impractical for real-time applications. Recent years have witnessed great progress in embedded and mobile devices like unmanned drones, smart phones, intelligent glasses, etc. The demand of deploying DNN models on these devices becomes more intensive. However, the resources of these devices, for example, the storage, computational units as well as the battery power are very limited, which pose a great challenge to accelerate modern DNNs in low-cost settings.

Therefore, how to equip specific hardware with efficient deep networks without significantly lowering the performance has become a critical problem. To deal with this issue, many great ideas and

[‡] Corresponding author

methods from the algorithm side have been investigated during the past few years. Some of these works may focus on model compression while others may focus on acceleration or lowering the power consumption. As for hardware side, a great variety of FPGA/ASIC-based accelerators are proposed for embedded and mobile applications. In this paper, we will give a comprehensive survey of several advanced approaches in network compression, acceleration and accelerator design. We will present the central ideas behind each approaches, giving the similarity and differences between different methods. Finally, we will give some future directions in this area.

The remaining part of this paper is organised as follows. In Section 2, we give the background of network acceleration and compression. From Section 3 to Section 7, we systematically describe a serial of hardware-efficient DNN algorithms, including network pruning, low-rank approximation, network quantization, teacher-student networks and compact network design. In Section 8, we introduce the design and implementation of hardware accelerators based on FPGA/ASIC. In Section 9, we make a discussion and present some future directions, and Section 10 concludes this paper.

2 Background

Recently deep convolutional neural networks have been quite popular due to their powerful representation capacity. With the huge success of CNNs, the demand of deploying deep networks to real world applications is increasing. However, the huge parameter size and computational complexity are the two key problems for the networks deployment. For the training phase of CNNs, the computational complexity is not a critical problem thanks to the high performance GPUs or CPU clouds. The parameter size also has little effect on the training phase since modern computers have very large disk and memory storage. But things are quite different for the inference phase of CNNs, especially on embedded and mobile devices.

The huge computational complexity brings two problems for deploying CNNs to real-world applications. One is that the inference phase of CNNs becomes slow as computational complexity is large. This makes it difficult to deploy CNNs to real-time applications. The other one is that the dense computation of CNNs will consume much battery power which is limited on mobile devices.

The large parameter size of CNNs mainly consume much storage and run-time memory which are quite limited on embeded devices. In addition, the large parameter size makes it difficult to download new models online on mobile devices.

To solve these problems, network compression and acceleration methods are proposed. In general, the computational complexity of CNNs is dominated by the convolutional layers while the number of parameters is mainly related to the full-connected layers as shown in Table 1. Thus most network acceleration methods focus on decreasing the computational complexity of convolutional layers, while the network compression methods mainly try to compress the fully-connected layers.

3 Network Pruning

Pruning method was proposed before deep learning becomes popular, and it has been widely studied in recent years (LeCun *et al.*, 1989; Hassibi and Stork, 1993; Han *et al.*, 2015a,b). Based on the assumption that lots of parameters in deep networks are unimportant or unnecessary, pruning methods remove unimportant parameters. In this way, pruning methods enlarge parameters' sparsity significantly. The high sparsity of parameters after pruning brings two benefits for deep neural networks. On the one hand, the sparse parameters after pruning require less disk storage since the parameters can be stored in the Compressed Sparse Row format (CSR) or Compressed Sparse Column (CSC) format. On the other hand, computations involved with those pruned parameters are omitted, thus the computational complexity of deep networks can be reduced. According to the granularity of pruning, pruning methods can be categorized into five groups: fine-grained pruning, vector-level pruning, kernel-level pruning, group-level pruning and filter-level pruning. Fig. 1 shows pruning methods with different granularities. In the following subsections, we give details of these different pruning methods.

3.1 Fine-grained Pruning

Fine-grained pruning methods or vanilla pruning methods remove parameters in an unstructured way, i.e., any unimportant parameters in the convo-

Table 1 The computation and parameter size of state-of-art convolution neural networks

Method	Parameters			Computation		
	Size(M)	Conv(%)	Fc(%)	FLOPS(G)	Conv(%)	Fc(%)
AlexNet	61	3.8	96.2	0.72	91.9	8.1
VGG-S	103	6.3	93.7	2.6	96.3	3.7
VGG16	138	10.6	89.4	15.5	99.2	0.8
NIN	7.6	100	0	1.1	100	0
GoogLeNet	6.9	85.1	14.9	1.6	99.9	0.1
ResNet-18	5.6	100	0	1.8	100	0
ResNet-50	12.2	100	0	3.8	100	0
ResNet-101	21.2	100	0	7.6	100	0

lutional kernels can be pruned, as shown in Fig. 1. Since there is no extra constraints on the pruning patterns, parameters can be pruned with a large sparsity. Early works of pruning (LeCun *et al.*, 1989; Hassibi and Stork, 1993) used the approximate second order derives of loss function w.r.t the parameters to determine the saliency of parameters, and then pruned those parameters with low saliency. But it's unaffordable for deep networks to compute the second order derives due to the huge computational complexity. Recently Han *et al.* (2015a) proposed the deep compression framework to compress deep neural networks with three steps: pruning, quantization, Huffman encoding. By using this method, AlexNet can be compressed by 35× without accuracy drops. After pruning, the pruned parameters in (Han *et al.*, 2015a) stay unchanged, thus those incorrect pruning may cause accuracy drops . To solve this problem, Guo *et al.* (2016) proposed the dynamic network surgery framework which consists of two operations: pruning and splicing. The pruning operation aims to prune those unimportant parameters while the splicing operation aims to recover those incorrect pruned connections. Their method requires less training epochs and achieves better compression ratio than (Han *et al.*, 2015a).

3.2 Vector-level and Kernel-level Pruning

Vector-level pruning methods prune vectors in the convolutioal kernels, and kernel-level pruning methods prune 2-D covolutional kernels in the filters. Since most pruning methods focus on fine-grained pruning and filter-level pruning, there are few works on vector-level and kernel-level pruning. Anwar *et al.* (2017) first explored the kernel-level pruning, and they also proposed a intra-kernel strided pruning method which prunes a sub-vector in a fixed stride. Mao *et al.* (2017) explored different level granularities of pruning, and they found that vector-level pruning takes less storage than fine-grained pruning because vector-level pruning requires fewer indices to indicate those pruned parameters. Besides, vector-level, kernel-level, and filter-level pruning are more friendly to hardware implementations since they are more structured pruning methods.

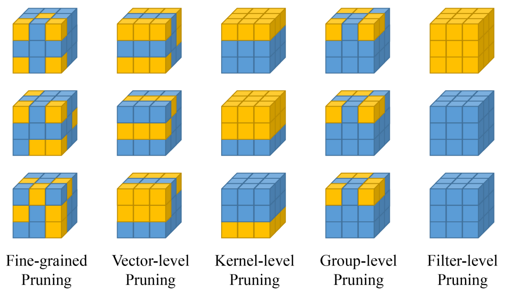

Fig. 1 Different Level Pruning Methods for a convolutional layer which has 3 convolutional filters with size $3 \times 3 \times 3$.

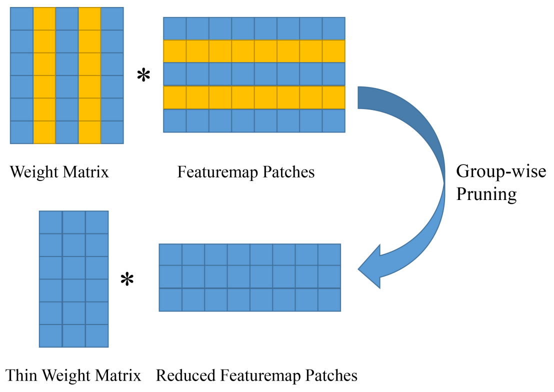

Fig. 2 Group-level Pruning.

3.3 Group-level Pruning

Group-level pruning methods prune the parameters by the same sparse pattern on the filters. As shown in Fig. 2, each filter has the same sparsity pattern, thus the convolutional filters can be represented as a thinned dense matrix. By using group-level pruning, convolutions can be implemented by thinned dense matrices multiplication. As a result, the Basic Linear Algebra Subprograms (BLAS) can be utilized to achieve a higher speed-up. Lebedev and Lempitsky (2016) proposed the Group-wise Brain Damage which prunes the weight matrix in a group-wise fashion. By using group-sparsity regularizers, deep networks can be trained easily with group-sparsified parameters. Since group-level pruning can utilize BLAS library, practical speed-up is almost linear in the sparsity level. By using this method, they achieved 3.2× speed-up for all convolutional layers of AlexNet. Concurrent with (Lebedev and Lempitsky, 2016), Wen *et al.* (2016) proposed to use the group Lasso to prune groups of parameters. Differently, Wen *et al.* (2016) explored different levels of structured sparsity in terms of filters, channels, filter shapes, and depth. Their method can be regarded as a more general group-regularized pruning method. For AlexNet's convolutional layers, their method achieves about 5.1× and 3.1× speed-ups on CPU and GPU respectively.

3.4 Filter-level Pruning

Filter-level pruning methods prune the convolutional filters or channels which makes the deep networks thinner. After pruning filters for one layer, the next layers' channels are also pruned. Thus, filter-level pruning is more efficient for accelerating the deep networks. Luo *et al.* (2017) proposed a filter-level pruning method named ThiNet. They used the next layer's feature map to guide the filter pruning in current layer. By minimizing the feature map's reconstruction error, they select the channels in a greedy way. Similar to (Luo *et al.*, 2017), He *et al.* (2017) proposed an iterative two-step algorithm to prune filters by minimizing the feature map errors. Specifically, they introduced a selection weight β_i for each filter $\mathbf{W_i}$, then added sparse constraints on β_i. Then the channel selection problem can be a LASSO regression problem. To minimize the feature map errors, they iteratively updated β and $\mathbf{W}$. And their method achieved 5× speed-up on VGG-16 network with little accuracy drop. Instead of using additional selection weight β, Liu *et al.* (2017) proposed to leverage the scaling factor of batch normlization layer for evaluating the importance of filters. By pruning channels with near-zero scaling factors, they can prune filters without introducing overhead to the networks.

4 Low-rank Approximation

The convolutional kernel of a convolutional layer $W \in R^{w \times h \times c \times n}$ is a 4-D tensor. These four dimensions correspond to the kernel width, kernel height, the number of input and output channels respectively. Note that by merging some of the dimensions, the 4-D tensor can be transformed into t-D $(t = 1, \cdots 4)$ tensor. The motivation behind low-rank decomposition is to find an approximate tensor $\hat{W}$ that is close to W but facilitates more efficient computation. Many low-rank based methods are proposed by the community, two key differences is how to rearrange the four dimensions, and on which dimension the low-rank constraint is imposed. Here we roughly divide the low-rank based methods into three categories, according to how many components the filters are decomposed into: two-component decomposition, three-component decomposition and four-component decomposition.

4.1 Two-component Decomposition

For two-component decomposition, the weight tensor is divided into two parts and the convolutional layer is replaced by two successive layers. Jaderberg *et al.* (2014) decomposed the spatial dimension $w * h$ into $w * 1$ and $1 * h$ filters. They achieved 4.5× speedup for a CNN trained on text character recognition dataset, with 1% accuracy drop.

SVD is a popular low-rank matrix decomposition method. By merging the dimensions w, h and c, the kernel becomes a 2-D matrix of size $(w*h*c) \times n$, on which the SVD decomposition method can be conducted on. In Denil *et al.* (2013), the authors utilized SVD to reduce the network redundancy. The SVD decomposition was also investigated in Zhang *et al.* (2015), in which the filters were replaced by two filter banks: one consists of d filters of shape $w \times h \times c$ and the other is composed of n filters of shape $1 \times 1 \times d$ filter. Here d represents the rank of the decomposition, i.e., the n filters are linear combinations of the first d filters. They also proposed the non-linear response reconstruction method based on the low-rank decomposition. On the challenging VGG-16 model for ImageNet classification task, this two-component SVD decomposition method achieved $3\times$ theoretical speedup at the cost of about 1.66% increased top-5 error.

Similarly, another SVD decomposition method can be used, by exploring the low-rank property along the input channel dimension c. In this way, we reshape the weight tensor into a matrix of size $c \times (w * h * n)$. By selecting the rank to d, the convolution can be decomposed by first a $1 \times 1 \times c \times d$ convolution and then a $w \times h \times d \times n$ convolution. These two decomposition are symmetric.

4.2 Three-component Decomposition

Based on the analysis of the two-component decomposition methods, one straightforward three-component decomposition method can be obtained by two successive two-component decomposition. Note that in the SVD decomposition, two weight tensors are introduced. The first is a $w \times h \times c \times d$ tensor and the other is a $d \times n$ tensor (matrix). The first convolution is also very time-consuming due to the large size of the first tensor. We can also conduct a two-component decomposition on the first weight tensor after the SVD decomposition, which turning into a three-component decomposition method. This strategy was studied in Zhang *et al.* (2015), in which after the SVD decomposition, they utilized the decomposition method proposed by Jaderberg *et al.* (2014) for the first decomposed tensor. Thus the final three components were convolutions with spatial size of $w \times 1$, $1 \times h$ and 1×1 respectively. By utilizing this three-component decomposition, only 0.3% increased top-5 error was achieved in (Zhang *et al.*, 2015) for $4\times$ theoretical speedup.

If we use the SVD decomposition along the input channel dimension for the first tensor after the two-component decomposition, we can get the Tucker decomposition format as proposed by Kim *et al.* (2015). These three components are convolutions of spacial size 1×1, $w \times h$ and another 1×1 convolution. Note that instead of using the two-step SVD decomposition, Kim *et al.* (2015) utilized the Tucker decomposition method directly to obtain these three components. Their method achieved $4.93\times$ theoretical speedup at the cost of 0.5% increased top-5 accuracy.

To further lower the complexity, Wang and Cheng (2016) proposed a Block-Term Decomposition (BTD) method based on low-rank and group sparse decomposition. Note that in the Tucker decomposition, the second component corresponding to the $w \times h$ convolution also needs a large number of computations. Because the second tensor is already low-rank along both the input and output channel dimensions, the decomposition methods discussed above can not be used any longer. Wang and Cheng (2016) proposed to approximate the original weight tensor by the sum of some smaller subtensors, each of which is in the Tucker decomposition format. By rearranging these subtensors, the BTD can be seen as a Tucker decomposition where the second decomposed tensor is a block diagonal tensor. By using this decomposition, they achieved 7.4% actual speedup for the VGG-16 model, at the cost of 1.3% increased on top-5 error.

4.3 Four-component Decomposition

By exploring the low-rank property along input/output channel dimension as well as the spatial dimension, the four-component decomposition can be obtained. This is corresponds to the CP-decomposition accelerating method proposed in Lebedev *et al.* (2014). In this way, the four components are convolutions of size 1×1, $w \times 1$, $1 \times h$ and 1×1. The CP-decomposition can achieve very high speedup ratio, however, due to the approximate error, only the second layer of AlexNet was processed in Lebedev *et al.* (2014). They achieved $4.5\times$ for the second layer of AlexNet at the cost of about 1% accuracy drop.

5 Network Quantization

Quantization is a approach for many compression and acceleration applications. It has a wide applications in image compression, information retrieval, etc. Many quantization methods are also investigated for network acceleration and compression. We mainly categorize these methods into two groups: scalar and vector quantization, which may need a codebook for quantization, and fixed-point quantization.

5.1 Scalar and Vector Quantization

Scalar and vector quantization technique has a long history, and it was original used for data compression. By using scalar or vector quantization, the original data can be represented by a codebook and a set of quantization codes. The codebook contains a set of quantization centers, and the quantization codes are used to indicate the assignment of quantization centers. In general, the number of quantization centers is far smaller than the number of original data. Besides, quantization codes can be encoded by lossless encoding method e.g. Huffman coding, or just represented as low-bit fixed points. Thus scalar or vector quantization can achieve high compression ratio. Gong *et al.* (2014) explored scalar and vector quantization techniques for compressing deep networks. For scalar quantization, they used the well-known K-means algorithm to compress the parameters. In addition, product quantization algorithm (PQ) (Jegou *et al.*, 2011), a special case of vector quantization, was leveraged to compress the fully-connected layers. By partitioning the feature space into several disjoint subspaces and then conducting K-means in each subspace, PQ algorithm can compress the fully-connected layers with little loss. As (Gong *et al.*, 2014) only compressed the fully-connected layers, Wu *et al.* (2016) proposed to utilize PQ algorithm to simultaneously accelerate and compress convolutional neural networks. They proposed to quantize the convolutional filters layer by layer via minimizing the feature map's reconstruction loss. During the inference phase, a look-up table is built by pre-computing the inner product between feature map patches and codebooks, then the output feature map can be calculated by simply accessing the look-up table. By using this method, they can achieve $4 \sim 6\times$ speed-up and $15 \sim 20\times$ compression ratio with little accuracy drop.

5.2 Fixed-point Quantization

Fixed-point quantization is an effective approach for lowering resource consumption of networks. Based on which part is quantized, two main categories can be classified, i.e., weight quantization and activation quantization. There are some other works that try to also quantize gradients, which can result in acceleration at the network traning stage. We maily review weight quantization and activation quantization methods, which accelerate the test-phase computation. Table 2 summarizes these methods according to which part is quantized and whether training and testing stage can be accelerated.

5.2.1 Fixed-point Quantization of Weights

Fixed-point weight quantization has been an old topic for network acceleration and compression. Hammerstrom (2012) proposed a VLSI architecture for network acceleration using 8-bit input and output and 16-bit internal representation. Later in Holi and Hwang (1993), the authors provided a theoretical analysis of error caused by low-bit quantization to determine the bit-width for a multilayer perceptron. They showed that 8-16 bit quantization was sufficient for training small neural networks. These early works mainly focused on simple multilayer perceptron. A more recent work (Chen *et al.*, 2014) showed that it is necessary to use 32-bit fixed-point for the convergence of a convolutional neural network trained on MNIST dataset. By using stochastic rounding, the work (Gupta *et al.*, 2015) found that it is sufficient to use 16-bit fixed-point numbers to train a convolutional neural network on MNIST. 8-bit fixed-point quantization was also investigated in Dettmers (2015) to speed up the convergence of deep networks in the parallel training. Logarithmic data representation was also investigated in Miyashita *et al.* (2016).

Recently, much lower bit quantization or even binary and ternary quantization methods were investigated. Expectation Backpropagation (EBP) was introduced in Cheng *et al.* (2015), which utilized variational Bayes method to binarize the network. The BinaryConnect method proposed in Courbariaux *et al.* (2015) constrained all weights to be either $+1$ or -1. By training from scratch, the BinaryConnect can even outperform the floating-point counterpart on CIFAR-10 (Krizhevsky and Hinton, 2009) image classification dataset. Using binary quantization, the network can be compressed by about 32 times compared to 32-bit floating-point networks. Most of the floating-point multiplication can also be eliminated (Lin *et al.*, 2015). In Rastegari *et al.* (2016), the authors proposed the Binary Weight Network (BWN), which was among the earliest works that achieved good results on large dataset of ImageNet (Russakovsky *et al.*, 2015). Loss-aware binarization was proposed in (Hou *et al.* (2016)), which can directly

Table 2 Fixed-point quantization methods comparison according to which part is quantized and whether training and testing stage can be accelerated.

Method	Quantization			Acceleration	
	Weight	Activation	Gradient	Training	Testing
BinaryConnect (Courbariaux *et al.*, 2015)	Binary	Full	Full	No	Yes
BWN (Rastegari *et al.*, 2016)	Binary	Full	Full	No	Yes
BWNH (Hu *et al.*, 2018)	Binary	Full	Full	No	Yes
TWN (Li *et al.*, 2016)	Binary	Full	Full	No	Yes
FFN (Wang and Cheng, 2017)	Ternary	Full	Full	No	Yes
INQ (Zhou *et al.*, 2017)	Ternary-5bit	Full	Full	No	Yes
BNN (Rastegari *et al.*, 2016)	Binary	Binary	Full	No	Yes
XNOR (Rastegari *et al.*, 2016)	Binary	Binary	Full	No	Yes
HWGQ (Cai *et al.*, 2017)	Binary	2bit	Full	No	Yes
DoReFa-Net (Zhou *et al.*, 2016)	Binary	1-4bit	6bit, 8bit, Full	Yes	Yes

minimize the classification loss with respect to the binarized weights. In the work of Hu *et al.* (2018), the authors proposed a novel approach named BWNH to train Binary Weight Networks via Hashing, which outperformed other weight binarization methods by a big margin. Ternary quantization was also utilized in Hwang and Sung (2014). In Li *et al.* (2016), the authors proposed Ternary Weight Network (TWN) which was similar BWN, but constrained all weights to be ternary values among {-1, 0, +1}. The TWN outperformed BWN by a large magin on deep models like ResNet. Trained Ternary Quantization proposed in Zhu *et al.* (2016a) learned both ternary values and ternary assignments at the same time using back-propagation. They achieved comparable results on AlexNet model. Different from previous quantization methods, the Incremental Network Quantization (INQ) method proposed in Zhou *et al.* (2017) gradually turned all weights into logarithmic format in a multi-step manner. This incremental quantization strategy can lower the quantization error during each stage, thus can make the quantization problem much easier. All these low-bit quantization methods discussed above directly quantize the full-precision weight into fixed-point format. In Wang and Cheng (2017), the authors proposed a very different quantization strategy. In stead of direct quantizaiton, they proposed to use a Fixed-point Factorization network (FFN) to quantize all weights into ternary values. This fixed-point decomposition method can significantly lower the quantization error. The FFN method achieved comparable results on commonly used deep models such as AlexNet, VGG-16 and ResNet.

5.2.2 Fixed-point Quantization of Activations

With only weight quantization, there is also the need of the time-consuming floating-point operations. If the activations were also quantized into fixed-point values, the network can be efficiently executed by only fixed-point operations. Many activation quantization methods were also proposed by the deep learning community. Bitwise neural network was proposed in Kim and Smaragdis (2016). Binarized Neural Networks (BNN) was among the first works that quantize both weights and activations into either -1 or +1. BNN achieved comparable accuracy with the full-precision baseline on CIFAR-10 dataset. To extend the BNN for ImageNet classification task, the authors of (Tang *et al.*, 2017) improved the training strategies of BNN. Much higher accuracy was reported using these strategies. Based on the BWN, the authors of (Rastegari *et al.*, 2016) further quantize all activations into binary values, making the network into XNOR-net. Compared with BNN, the XNOR-net can achieve much higher accuracy on ImageNet dataset. To further understand the effect of bit-with on the training of deep neural networks, the DoReFa-net was proposed in Zhou *et al.* (2016). It investigated the effect of different bit-with for weights, activations as well as gradients. By making use of batch normalization, the work (Cai *et al.*, 2017) presented the Half-wave Gaussian Quantization (HWGQ) method to quantize both weights and activations. High performance was achieved on commonly used CNN models using HWGQ methond, with 2-bit activations and binary weights.

6 Teacher-student Network

Teacher-student network is different with those network compression or acceleration methods since it trains a student network by a teacher network and the student network can be designed with totally different network architecture. Generally speaking, a teacher network is a large neural network or the ensemble of neural networks while a student network is a compact and efficient neural network. By utilizing the dark knowledge transferred from the teacher network, the student network can achieve higher accuracy than training merely by the class labels. Hinton *et al.* (2015) proposed the knowledge distillation (KD) method which trains a student network by the softmax layer's output of teacher network. Following this line, Romero *et al.* (2014) proposed the FitNets to train a deeper and thinner student network. Since the depth of neural networks is more important than the width of them, a deeper student network would have higher accuracy. Besides, they utilized the both intermediate layers' featuremaps and soft outputs of teacher network to train the student network. Rather than mimicing the intermediate layers' featuremaps, Zagoruyko and Komodakis (2016) proposed to train a student network by imitating the attention maps of a teacher network. Their experiments showed that the attention maps are more important than layers' activations and their method can achieve higher accuracy than FitNets.

7 Compact Network Design

The objective of network acceleration and compression is to optimize the executation and storage framework for a given deep neural network. One property is that the network architecture is not changed. Another parallel branch for network acceleration and compression is to design more efficient but low-cost network architecture itself.

In Lin *et al.* (2013), the authors proposed Network-In-Network architecture, where 1×1 convolution were utilized to increase the network capacity while keep the overall computational complexity small. To reduce the storage requirement of the CNN models, they also proposed to remove the fully-connected layer and make use of a global average pooling. These strategies are also used by many state-of-the-art CNN models like GoogleNet (Szegedy *et al.*, 2015) and ResNet (He *et al.*, 2016).

Branching (mutiple group convolution) is another commonly used strategy for lowering network complexity, which was explored in the work of GoogleNet (Szegedy *et al.*, 2015). By largely making use of 1×1 convolution and the branching strategy, the SqueezeNet proposed in Iandola *et al.* (2016) achieved about 50$\times$ compression than AlexNet, with comparable accuracy. By branching, the work of ResNeXt (Xie *et al.*, 2017) can achieve much higher accuracy than the ResNet (He *et al.*, 2016) at the same computational budget. The depthwise convolution proposed in MobileNet (Howard *et al.*, 2017) takes the branching strategy to the extreme, i.e., the number of branches equals to the number of input/output channels. The resulting MobileNet can be 32$\times$ smaller and 27$\times$ faster than VGG-16 model, with comparable image classification accuracy on ImageNet. When using depthwise convolution and 1×1 convolution as in MobileNet, most of the computation and parameters reside in the 1×1 convolutions. On strategy to further lower the complexity of the 1×1 convolution is to use multiple groups. The ShuffleNet proposed in Zhang *et al.* (2017) introduced the channel shuffle operation to increase the information change within the multiple groups, which can prominently increase the representation power of the networks. Their method achieved about 13$\times$ actual speedup over AlexNet with comparable accuracy.

8 Hardware Accelerator

8.1 Background

Deep neural networks provide impressive performance on various tasks while suffering from huge computational complexity. Traditionally, algorithms based on deep neural networks should be executed on general purpose platforms such as CPUs and GPUs, but this work at the cost of unexpected power consumption and oversized resource utilization for both computing and storage. In recent years, there is an increasing number of applications that based on embedded systems, including autonomous vehicles, unmanned drones, security cameras, etc. Considering the demands of high performance, lightweight and low energy cost on these devices, CPU/GPU-based solutions are no longer suitable. In this scenario,

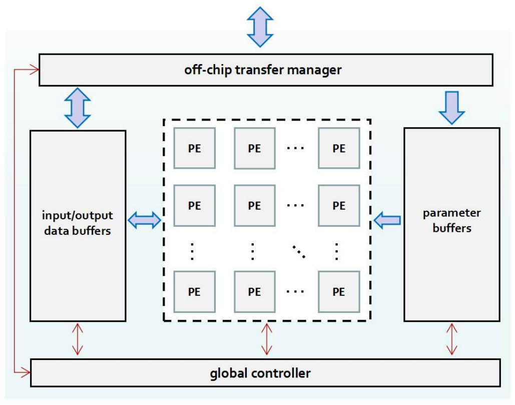

Fig. 3 General architecture of an accelerator on dedicated hardware.

FPGA/ASIC-based hardware accelerators are gaining popularity as efficient alternatives.

8.2 General Architecture

The deployment of DNN on a real-world application consists of two phases: training and inference. Network training is known to be expensive in terms of speed and memory, thus is usually carried on GPUs off-line. During inference, the pre-trained network parameters can be loaded either from cloud or from dedicated off-chip memory. Most recently, hardware accelerators for training are given attentions (Ko *et al.*, 2017; Yang, 2017; Venkataramani *et al.*, 2017), but in this section we mainly focus on the inference phase in embedded settings.

Typically, an accelerator is composed of five parts: data buffers, parameter buffers, processing elements, global controller, off-chip transfer manager, as shown in Fig. 3. The data buffers are used to caching input images, intermediate data and output predictions, while the weight buffers are mainly used to caching convolutional filters. Processing elements are a collection of basic computing units that executing multiply-adds, non-linearity and any other functions such as normalization, quantization, etc. The global controller is used to orchestrate the computing flow on chip, while off-chip transfers of data and instructions are conducted through a manager. This basic architecture can be found in existing accelerators designed for both specific and general tasks.

Heterogeneous computing is widely adopted in hardware acceleration. For computing-intensive operations such as multiply-adds, it is efficient to fit them on hardware for high throughout, otherwise, data pre-processing, softmax and any other graphic operations can be placed on CPU/GPU for low latency processing.

8.3 Processing Elements

Among all of the accelerators, the biggest differences exist in processing elements as they are designed for the the majority computing tasks in deep networks, such as massive multiply-add operations, normalization (batch normalization or local response normalization), non-linearities (ReLU, sigmoid and tanh). Typically, the computing engine of an accelerator is composed of many small basic processing elements, as shown in Fig. 3, this architecture is mainly designed for fully invest data reuse and parallelism. However, there are a lot of accelerators that with only one processing element for the consideration of less data movements and resource saving (Zhang, 2015; Ma *et al.*, 2017c).

8.4 Optimizing for High Throughput

Since the majority computations in network are matrix-matrix/matrix-vector multiplication, it is critical to deal with the massive nested loops to achieve high throughput. Loop optimization is one of the most frequently adopted techniques in accelerator design (Zhang, 2015; Ma *et al.*, 2017b; Suda, 2016; Alwani *et al.*, 2016; Xiao, 2017; Li *et al.*, 2018), including loop tiling, loop unrolling, loop interchange, etc. Loop tiling is used to divide the entire data into multiple small blocks in order to alleviate the pressure of on-chip storage (Ma *et al.*, 2017b; Alwani *et al.*, 2016; Qiu, 2016), while loop unrolling attempts to improve the parallelism of computing engine for high speed (Ma *et al.*, 2017b; Qiu, 2016). Loop interchange determines the sequential computation order of nested loops, since different orders can result in significantly difference performance. The well-known *systolic array* can be seen as a combination of the loop optimization methods listed above, which leverages the nature of data locality and weight sharing in network to achieve high throughput (Jouppi, 2017; Wei, 2017).

SIMD-based computation is another way for high throughput. Nguyen *et al.* (2017) presented a method for packing two low-bit multiplications into a single DSP block to double the computation, Price *et al.* (2017) also proposed a SIMD-based architecture for speech recognition.

8.5 Optimizing for Low Energy Consumption

Existing work attempt to reduce energy consumption of a hardware accelerator from both computing and IO perspectives. Horowitz (2014) systematically illustrated the energy cost in terms of arithmetic operations and memory accesses. It demonstrated that operations based on integer are much more cheap than float-point counterparts, and lower bit integer is better. Therefore, most existing accelerators adopt low-bit or even binary data representation (Zhao, 2017; Umuroglu, 2017; Nurvitadhi, 2017) to preserve energy efficiency. Most recently, logrithmic computation that transfers multiplications into bit shift operations has also shown its promise in energy saving (Edward, 2017; Gudovskiy and Rigazio, 2017; Tann *et al.*, 2017).

Sparsity is gaining an increasing popularity in accelerator design based on the observation that a great amount of arithmetic operations can be discarded to obtain energy efficiency. Han *et al.* (2016), Han *et al.* (2017) and Parashar *et al.* (2017) designed architectures for image or speech recognition based on network pruning, while Albericio *et al.* (2016) and Zhang *et al.* (2016c) proposed to eliminate ineffectual operations based on the inherent sparsity in networks.

Off-chip data transfers heavily exist in hardware accelerators due to the fact that both network parameters and intermediate data are too large to fit on chip. Horowitz (2014) suggested that power consumption caused by DRAM access is several orders of magnitude of SRAM, thus reducing off-chip transfers is an critical issue. Shen *et al.* (2017) designed a flexible data buffing scheme to reduce bandwidth requirements, Alwani *et al.* (2016) and Xiao (2017) proposed fusion-based method to reduce off-chip traffic. Most recently, Li *et al.* (2018) presented a block-based convolution that can completely avoid off-chip transfers of intermediate data in VGG-16 with high throughput.

Many other approaches have been proposed to reduce power consumption. Zhang *et al.* (2016b) used a pipelined FPGA cluster to realize acceleration, Chen *et al.* (2017b) presented an energy-efficient row stationary scheme to reduce data movements, Zhu *et al.* (2016b) attempted to reduce power consumption via low-rank approximation.

8.6 Design Automation

Recently, design automation frameworks that automatically mapping deep neural networks onto hardware gain an increasing attention. Wang *et al.* (2016), Sharma *et al.* (2016), Venieris and Bouganis (2016) and Wei (2017) proposed frameworks that automatically generate synthesizable accelerator for a given network. Ma *et al.* (2017a) presented an RTL compiler for FPGA implementation of diverse networks. Liu *et al.* (2016) proposed an instruction set for hardware implementation, while Zhang *et al.* (2016a) proposed a uniformed convolutional matrix multiplication representation for CNNs.

8.7 Emerging Techniques

In the past few years, there are a lot of new techniques both from algorithm side and circuit side are adopted to implement fast and energy-efficient accelerators. Stochastic computing that represents continuous values by streams of random bits are investigated for hardware acceleration of deep neural networks (Ren *et al.*, 2017; Sim and Lee, 2017; Kim *et al.*, 2016b). On the hardware side, RRAM-based accelerators (Chen *et al.*, 2017a; Xia *et al.*, 2016) and the usage of 3D DRAM (Kim *et al.*, 2016a; Gao *et al.*, 2017) are given more attentions.

9 The Trend and Discussion

In this section, we want to give some possible future directions in this topic.

Non-finetuning or Unsupervised Compression. Most of existing methods, including network pruning, low-rank compression and quantization, need labeled data to retrain the network for accuracy retaining. The problems are twofold. First, labeled data is sometimes unavailable, like the medical images. Another problem is that retraining needs a lot of human work as well as professional knowledge. These two problems raise the need for unsupervised compression or even finetuning-free compression methods.

Scalable (Self-adaptive) Compression. Current compression methods have lots of hyper parameters to be determined ahead. For example, the sparsity of network pruning, the rank of decomposition-based methods or the bit-width of fixed-point quantization methods. The selection of these hyper parameters is a tedious work, which also needs professional experiences. Thus the investigation of methods which do not rely on human-designed hyper parameters is a promising research topic. One direction may be using annealing methods, or reinforcement learning.

Network Acceleration for Object Detection. Most of model acceleration methods are optimized for image classification, yet very few efforts have devoted to accelerate other computer vision tasks such object detection. It seems that model acceleration methods for image classification can be directly used for detection. However, the deep neural networks for object detection or image segmentation are more sensitive to model acceleration methods, i.e using the same model acceleration methods for object detection would suffer from more accuracy drops than image classification. One reason for this phenomenon may be that object detection requires more complex feature representation than image classification. How to design model acceleration methods for object detection seems a challenge.

Hardware-software Co-design. To accelerate deep learning algorithms on dedicated hardware, a straightforward method is to pick up a model and design a corresponding architecture. However, the gap between algorithm modeling and hardware implementation will make it difficult to put in practice. Recent advances on deep learning algorithms and hardware accelerators demonstrate that it is highly desirable to design hardware-efficient algorithms according to low-level features of specific hardware platforms. This co-design methodology will be a trendcy in future work.

10 Conclusion

Deep neural networks provide impressive performance while suffering from huge computational complexity and energy cost. In this paper, we give a survey of recent advances in efficient processing of deep neural networks from both algorithm and hardware side. Besides, we point out a few of topics that deserve to look at in future.

References

Albericio, J., Judd, P., Hetherington, T., *et al.*, 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. International Symposium on Computer Architecture, p.1-13.

Alwani, M., Chen, H., Ferdman, M., *et al.*, 2016. Fused-layer cnn accelerators. MICRO.

Anwar, S., Hwang, K., Sung, W., 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, **13**(3):32.

Cai, Z., He, X., Sun, J., *et al.*, 2017. Deep learning with low precision by half-wave gaussian quantization, .

Chen, L., Li, J., Chen, Y., *et al.*, 2017a. Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar. Design, Automation and Test in Europe Conference and Exhibition, p.19-24.

Chen, Y.H., Krishna, T., Emer, J.S., *et al.*, 2017b. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, **52**(1):127-138.

Chen, Y., Sun, N., Temam, O., *et al.*, 2014. Dadiannao: A machine-learning supercomputer. Ieee/acm International Symposium on Microarchitecture, p.609-622.

Cheng, Z., Soudry, D., Mao, Z., *et al.*, 2015. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, .

Courbariaux, M., Bengio, Y., David, J.P., 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. Advances in Neural Information Processing Systems, p.3123-3131.

Denil, M., Shakibi, B., Dinh, L., *et al.*, 2013. Predicting parameters in deep learning. Advances in Neural Information Processing Systems, p.2148-2156.

Dettmers, T., 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, .

Edward, 2017. Lognet: Energy-efficient neural networks using logarithmic computation. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, :5900-5904.

Gao, M., Pu, J., Yang, X., *et al.*, 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. International Conference on Architectural Support for Programming Languages and Operating Systems, p.751-764.

Gong, Y., Liu, L., Yang, M., *et al.*, 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, .

Gudovskiy, D., Rigazio, L., 2017. ShiftCNN: Generalized low-precision architecture for inference of convolutional neural networks. *arXiv preprint arXiv:1706.02393*, .

Guo, Y., Yao, A., Chen, Y., 2016. Dynamic network surgery for efficient dnns. Advances In Neural Information Processing Systems, p.1379-1387.

Gupta, S., Agrawal, A., Gopalakrishnan, K., *et al.*, 2015. Deep learning with limited numerical precision. Proceedings of the 32nd International Conference on Machine Learning (ICML-15), p.1737-1746.

Hammerstrom, D., 2012. A vlsi architecture for high-performance, low-cost, on-chip learning. IJCNN International Joint Conference on Neural Networks, p.537-544 vol.2.

Han, S., Mao, H., Dally, W.J., 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, .

Han, S., Pool, J., Tran, J., *et al.*, 2015b. Learning both weights and connections for efficient neural network. Advances in Neural Information Processing Systems, p.1135-1143.

Han, S., Liu, X., Mao, H., *et al.*, 2016. Eie: Efficient inference engine on compressed deep neural network. ACM/IEEE International Symposium on Computer Architecture, p.243-254.

Han, S., Kang, J., Mao, H., *et al.*, 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga, .

Hassibi, B., Stork, D.G., 1993. Second order derivatives for network pruning: Optimal brain surgeon. Morgan Kaufmann.

He, K., Zhang, X., Ren, S., *et al.*, 2016. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, .

He, Y., Zhang, X., Sun, J., 2017. Channel pruning for accelerating very deep neural networks. *arXiv preprint arXiv:1707.06168*, .

Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, .

Holi, J.L., Hwang, J.N., 1993. Finite precision error analysis of neural network hardware implementations. Ijcnn-91-Seattle International Joint Conference on Neural Networks, p.519-525 vol.1.

Horowitz, M., 2014. 1.1 computing's energy problem (and what we can do about it). Solid-State Circuits Conference Digest of Technical Papers, p.10-14.

Hou, L., Yao, Q., Kwok, J.T., 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600*, .

Howard, A.G., Zhu, M., Chen, B., *et al.*, 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, .

Hu, Q., Wang, P., Cheng, J., 2018. From hashing to cnns: Training binary weight networks via hashing. AAAI.

Hwang, K., Sung, W., 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. 2014 IEEE Workshop on Signal Processing Systems (SiPS), p.1-6.

Iandola, F.N., Han, S., Moskewicz, M.W., *et al.*, 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, .

Jaderberg, M., Vedaldi, A., Zisserman, A., 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, .

Jegou, H., Douze, M., Schmid, C., 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, **33**(1):117-128.

Jouppi, N.P., 2017. In-datacenter performance analysis of a tensor processing unit. Proceedings of the 44th Annual International Symposium on Computer Architecture.

Kim, D., Kung, J., Chai, S., *et al.*, 2016a. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. International Symposium on Computer Architecture, p.380-392.

Kim, K., Kim, J., Yu, J., *et al.*, 2016b. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. Design Automation Conference, p.124.

Kim, M., Smaragdis, P., 2016. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, .

Kim, Y.D., Park, E., Yoo, S., *et al.*, 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, .

Ko, J.H., Mudassar, B., Na, T., *et al.*, 2017. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. Design Automation Conference, p. 59.

Krizhevsky, A., Hinton, G., 2009. Learning multiple layers of features from tiny images, .

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, p.1097-1105.

Lebedev, V., Lempitsky, V., 2016. Fast convnets using group-wise brain damage. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, p.2554-2564.

Lebedev, V., Ganin, Y., Rakhuba, M., *et al.*, 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, .

LeCun, Y., Denker, J.S., Solla, S.A., *et al.*, 1989. Optimal brain damage. Advances in Neural Information Processing Systems, **89**.

Li, F., Zhang, B., Liu, B., 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, .

Li, G., Li, F., Zhao, T., *et al.*, 2018. Block convolution: Towards memory-efficeint inference of large-scale cnns on fpga. Design Automation and Test in Europe.

Lin, M., Chen, Q., Yan, S., 2013. Network in network. *arXiv preprint arXiv:1312.4400*, .

Lin, Z., Courbariaux, M., Memisevic, R., *et al.*, 2015. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, .

Liu, S., Du, Z., Tao, J., *et al.*, 2016. Cambricon: An instruction set architecture for neural networks. *SIGARCH Comput. Archit. News*, **44**(3).

Liu, Z., Li, J., Shen, Z., *et al.*, 2017. Learning efficient convolutional networks through network slimming. *arxiv preprint*, **1708**.

Luo, J.H., Wu, J., Lin, W., 2017. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, .

Ma, Y., Cao, Y., Vrudhula, S., *et al.*, 2017a. An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks. International Conference on Field Programmable Logic and Applications, p.1-8.

Ma, Y., Cao, Y., Vrudhula, S., *et al.*, 2017b. Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Ma, Y., Kim, M., Cao, Y., *et al.*, 2017c. End-to-end scalable fpga accelerator for deep residual networks. IEEE International Symposium on Circuits and Systems, p.1-4.

Mao, H., Han, S., Pool, J., *et al.*, 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, .

Miyashita, D., Lee, E.H., Murmann, B., 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, .

Nguyen, D., Kim, D., Lee, J., 2017. Double MAC: doubling the performance of convolutional neural networks on modern fpgas. Design, Automation and Test in Europe Conference and Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017, p.890-893.

Nurvitadhi, 2017. Can fpgas beat gpus in accelerating next-generation deep neural networks? Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Parashar, A., Rhu, M., Mukkara, A., *et al.*, 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks, :27-40.

Price, M., Glass, J., Chandrakasan, A.P., 2017. 14.4 a scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating. Solid-State Circuits Conference, p.244-245.

Qiu, 2016. Going deeper with embedded fpga platform for convolutional neural network. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Rastegari, M., Ordonez, V., Redmon, J., *et al.*, 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. ECCV (4), **9908**:525-542.

Ren, A., Li, Z., Ding, C., *et al.*, 2017. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *Acm Sigops Operating Systems Review*, **51**(2):405-418.

Romero, A., Ballas, N., Kahou, S.E., *et al.*, 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, .

Russakovsky, O., Deng, J., Su, H., *et al.*, 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, **115**(3):211-252.

Sharma, H., Park, J., Mahajan, D., *et al.*, 2016. From high-level deep neural models to fpgas. Ieee/acm International Symposium on Microarchitecture, p.1-12.

Shen, Y., Ferdman, M., Milder, P., 2017. Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer. IEEE International Symposium on Field-Programmable Custom Computing Machines.

Sim, H., Lee, J., 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. Design Automation Conference, p. 29.

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

Suda, 2016. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Szegedy, C., Liu, W., Jia, Y., *et al.*, 2015. Going deeper with convolutions. Computer Vision and Pattern Recognition, p.1-9.

Tang, W., Hua, G., Wang, L., 2017. How to train a compact binary neural network with high accuracy? AAAI, p.2625-2631.

Tann, H., Hashemi, S., Bahar, I., *et al.*, 2017. Hardware-software codesign of accurate, multiplier-free deep neural networks. *CoRR*, **abs/1705.04288**.

Umuroglu, 2017. Finn: A framework for fast, scalable binarized neural network inference. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Venieris, S.I., Bouganis, C.S., 2016. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. IEEE International Symposium on Field-Programmable Custom Computing Machines, p.40-47.

Venkataramani, S., Ranjan, A., Banerjee, S., *et al.*, 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. *SIGARCH Comput. Archit. News*, **45**(2):13-26.

Wang, P., Cheng, J., 2016. Accelerating convolutional neural networks for mobile applications. Proceedings of the 2016 ACM on Multimedia Conference, p.541-545.

Wang, P., Cheng, J., 2017. Fixed-point factorized networks. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Wang, Y., Xu, J., Han, Y., *et al.*, 2016. Deepburning: automatic generation of fpga-based learning accelerators for the neural network family. Design Automation Conference, p.110.

Wei, 2017. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. Proceedings of the 54th Annual Design Automation Conference 2017.

Wen, W., Wu, C., Wang, Y., *et al.*, 2016. Learning structured sparsity in deep neural networks. Advances in Neural Information Processing Systems, p.2074-2082.

Wu, J., Leng, C., Wang, Y., *et al.*, 2016. Quantized convolutional neural networks for mobile devices. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, .

Xia, L., Tang, T., Huangfu, W., *et al.*, 2016. Switched by input: Power efficient structure for rram-based convolutional neural network. Design Automation Conference, p.125.

Xiao, 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas. Proceedings of the 54th Annual Design Automation Conference 2017.

Xie, S., Girshick, R., Dollar, P., *et al.*, 2017. Aggregated residual transformations for deep neural networks. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Yang, H., 2017. Time: A training-in-memory architecture for memristor-based deep neural networks. Design Automation Conference, p. 26.

Zagoruyko, S., Komodakis, N., 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, .

Zhang, 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Zhang, C., Fang, Z., Pan, P., *et al.*, 2016a. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. International Conference on Computer-Aided Design, p. 12.

Zhang, C., Wu, D., Sun, J., *et al.*, 2016b. Energy-efficient cnn implementation on a deeply pipelined fpga cluster. Proceedings of the 2016 International Symposium on Low Power Electronics and Design.

Zhang, S., Du, Z., Zhang, L., *et al.*, 2016c. Cambricon-x: An accelerator for sparse neural networks. Ieee/acm International Symposium on Microarchitecture, p.1-12.

Zhang, X., Zou, J., He, K., *et al.*, 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, .

Zhang, X., Zhou, X., Lin, M., *et al.*, 2017. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, .

Zhao, 2017. Accelerating binarized convolutional neural networks with software-programmable fpgas. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

Zhou, A., Yao, A., Guo, Y., *et al.*, 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, .

Zhou, S., Wu, Y., Ni, Z., *et al.*, 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, .

Zhu, C., Han, S., Mao, H., *et al.*, 2016a. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, .

Zhu, J., Qian, Z., Tsui, C.Y., 2016b. Lradnn: High-throughput and energy-efficient deep neural network accelerator using low rank approximation. Asia and South Pacific Design Automation Conference, p.581-586.