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

Review:

Recent advances in efficient computation of deep convolutional neural networks

Jian CHENG[†‡1,2], Pei-song WANG[†1,2], Gang LI[†1,2], Qing-hao HU[†1,2], Han-qing LU[1,2]

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

[2]*University of Chinese Academy of Sciences, Beijing 100049, China*

[†]E-mail: jcheng@nlpr.ia.ac.cn; peisong.wang@nlpr.ia.ac.cn; gang.li@nlpr.ia.ac.cn; qinghao.hu@nlpr.ia.ac.cn

Received Nov. 25, 2017; Revision accepted Jan. 22, 2018; Crosschecked Jan 26, 2018

Abstract: Deep neural networks have evolved remarkably over the past few years and they are currently the fundamental tools of many intelligent systems. At the same time, the computational complexity and resource consumption of these networks continue to increase. This poses a significant challenge to the deployment of such networks, especially in real-time applications or on resource-limited devices. Thus, network acceleration has become a hot topic within the deep learning community. As for hardware implementation of deep neural networks, a batch of accelerators based on a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC) have been proposed in recent years. In this paper, we provide a comprehensive survey of recent advances in network acceleration, compression, and accelerator design from both algorithm and hardware points of view. Specifically, we provide a thorough analysis of each of the following topics: network pruning, low-rank approximation, network quantization, teacher–student networks, compact network design, and hardware accelerators. Finally, we introduce and discuss a few possible future directions.

Key words: Deep neural networks; Acceleration; Compression; Hardware accelerator
https://doi.org/10.1631/FITEE.1700789 **CLC number:** TP3

1 Introduction

In recent years, deep neural networks (DNNs) have achieved remarkable performance across a wide range of applications, including but not limited to computer vision, natural language processing, and speech recognition. These breakthroughs are closely related to the increased amount of training data and more powerful computing resources now available. For example, one breakthrough in the natural image recognition field was achieved by AlexNet (Krizhevsky et al., 2012). It was trained using multiple graphics processing units (GPUs) on about 1.2 M images. Since then, the performance of DNNs has continued to improve. For many tasks, DNNs are reported to be able to outperform humans. The problem, however, is that the computational complexity, and the storage requirements of these DNNs, have also increased drastically as shown in Table 1. Specifically, the widely used VGG-16 model (Simonyan and Zisserman, 2014) involves a storage of more than 500 MB and over 15 G floating-point operations (FLOPs) to classify a single 224 × 224 image.

Thanks to the recent crop of powerful GPUs and central processing unit (CPU) clusters equipped with more abundant memory resources and computational units, these more powerful DNNs can be trained within a relatively reasonable time period. However, for the inference phase, such a long execution time is impractical for real-time

‡ Corresponding author
ORCID: Jian CHENG, http://orcid.org/0000-0003-1289-2758
© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2018

applications. Recent years have witnessed a great progress in embedded and mobile devices including unmanned drones, smart phones, intelligent glasses, etc. The demand for deployment of DNN models on these devices has become intense. However, the resources of these devices, for example, the storage and computational units as well as the battery power, remain limited, and this poses a real challenge in accelerating modern DNNs in low-cost settings.

Therefore, a critical current problem is how to equip specific hardware with efficient deep networks without significantly lowering the performance. To deal with this issue, many great ideas and methods from the algorithm side have been investigated over the past few years. Some of the work focuses on model compression while others focus on acceleration or lowering power consumption. As for the hardware side, a wide variety of field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC) based accelerators have been proposed for embedded and mobile applications. In this paper, we present a comprehensive survey of several advanced approaches in network compression, acceleration, and accelerator design. We will present the central ideas behind each approach and explore the similarities and differences among the different methods. Finally, we will present some future directions in the field.

2 Background

Recently, deep convolutional neural networks (CNNs) have become quite popular due to their powerful representational capacity. With the huge success of CNNs, the demand for deployment of deep networks in real world applications has been increasing. However, the large storage consumption and computational complexity remain two key problems for deployment of these networks. For the CNN training phase, the computational complexity is not a critical problem, thanks to the high-performance GPUs or CPU clouds. The large storage consumption also has less effect on the training phase because modern computers have very large disks and memory storage capacities. However, things are quite different for the inference phase in CNNs, especially with regard to embedded and mobile devices.

The enormous computational complexity introduces two problems in the deployment of CNNs in real-world applications. One is that the CNN inference phase slows down as the computational complexity grows. This makes it difficult to deploy CNNs in real-time applications. The other problem is that the dense computation inherent to CNNs will consume substantial battery power, which is limited on mobile devices.

The parameters of CNNs consume a considerable storage and a run-time memory, which are quite limited on embedded devices. In addition, it becomes difficult to download new models online on mobile devices.

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

3 Network pruning

Pruning methods were proposed before deep learning became popular, and they have been studied widely in recent years (LeCun et al., 1989; Hassibi and Stork, 1993; Han et al., 2015a,b). Based on the assumption that many parameters in deep networks are unimportant or unnecessary, pruning methods are used to remove these parameters. In this way, pruning methods can expand the sparsity of the parameters significantly. The high sparsity of the parameters after pruning introduces two benefits for DNNs. On the one hand, the sparse parameters after pruning require less disk storage since the parameters can be stored in the compressed sparse row (CSR) format or compressed sparse column (CSC) format. On the other hand, computations involving these 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 the pruning methods with different granularities. In Sections 3.1–3.4, we will describe the different pruning methods in detail.

Table 1 Computation and parameters for state-of-the-art convolution neural networks

Method	Parameter			Computation		
	Size (M)	Conv (%)	Fc (%)	FLOPs (G)	Conv (%)	Fc (%)
AlexNet	61.0	3.8	96.2	0.72	91.9	8.1
VGG-S	103.0	6.3	93.7	2.60	96.3	3.7
VGG16	138.0	10.6	89.4	15.50	99.2	0.8
NIN	7.6	100.0	0	1.10	100.0	0
GoogLeNet	6.9	85.1	14.9	1.60	99.9	0.1
ResNet-18	5.6	100.0	0	1.80	100.0	0
ResNet-50	12.2	100.0	0	3.80	100.0	0
ResNet-101	21.2	100.0	0	7.60	100.0	0

FLOPs: floating-point operations; Conv: convolutional layers; Fc: fully-connected layers

3.1 Fine-grained pruning

Fine-grained or vanilla pruning methods remove parameters in an unstructured way; i.e., any unimportant parameter in the convolutional kernels can be pruned, as shown in Fig. 1. Since there are no extra constraints on the pruning patterns, the parameters can be pruned with a high sparsity. Early work on pruning (LeCun et al., 1989; Hassibi and Stork, 1993) used the approximate second-order derivatives of the loss function w.r.t. the parameters to determine the saliency of the parameters, and then pruned those parameters with a low saliency. Yet, deep networks can ill afford to compute the second-order derivatives due to the huge computational complexity. Recently, Han et al. (2015a) proposed a deep compression framework to compress DNNs in three steps: pruning, quantization, and Huffman encoding. By using this method, AlexNet could be compressed by 35-fold without drops in accuracy. After pruning, the pruned parameters in Han et al. (2015a) remain unchanged, and incorrectly pruned parameters could cause accuracy drops. To solve this problem, Guo et al. (2016) proposed a 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 the incorrectly pruned connections. Their method requires fewer training epochs and achieves a better compression ratio than that of Han et al. (2015a).

3.2 Vector-level and kernel-level prunings

Vector-level pruning methods prune vectors in the convolutional kernels, and kernel-level pruning methods prune 2D convolutional kernels in the filters. Since most pruning methods focus on fine-grained pruning or filter-level pruning, there is little work on vector- and kernel-level prunings. Anwar et al. (2017) first explored kernel-level pruning, and then proposed an intra-kernel strided pruning method, which prunes a sub-vector in a fixed stride. Mao et al. (2017) explored different granularity levels in pruning, and found that vector-level pruning takes up less storage than fine-grained pruning because vector-level pruning requires fewer indices to indicate the pruned parameters. Nevertheless, vector-, kernel-, and filter-level pruning techniques are more efficient in hardware implementations since they are friendlier to the memory access than non-structured pruning methods.

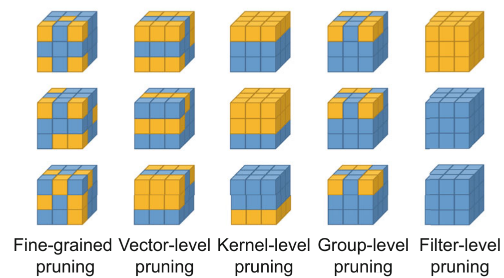

Fig. 1 Different pruning methods for a convolutional layer which has three convolutional filters of size 3 × 3 × 3

3.3 Group-level pruning

Group-level pruning methods prune the parameters according to the same sparse pattern on the filters. As shown in Fig. 2, each filter has the same sparsity pattern, and thus the convolutional filters can be represented as a thinned dense matrix. By using group-level pruning, convolutions can be implemented by thinned dense matrix multiplication. As a result, the basic linear algebra

subprograms (BLAS) can be used to achieve a higher speed-up. Lebedev and Lempitsky (2016) proposed the group-wise brain damage approach, which prunes the weight matrix in a group-wise fashion. By using group-sparsity regularization, deep networks can be trained easily with group-sparsified parameters. Since group-level pruning can use the BLAS library, the practical speed-up is almost linear at the sparsity level. By using this method, they achieved a 3.2-fold speed-up for all convolutional layers in AlexNet. Concurrent with Lebedev and Lempitsky (2016), Wen et al. (2016) proposed using the group Lasso to prune groups of parameters. In contrast, Wen et al. (2016) explored different levels of structured sparsity in terms of filters, channels, filter shapes, and depth. Their methods can be regarded as more general group-regularized pruning methods. For AlexNet's convolutional layers, Wen et al. (2016) achieved about 5.1- and 3.1-fold speed-ups on a CPU and GPU, respectively.

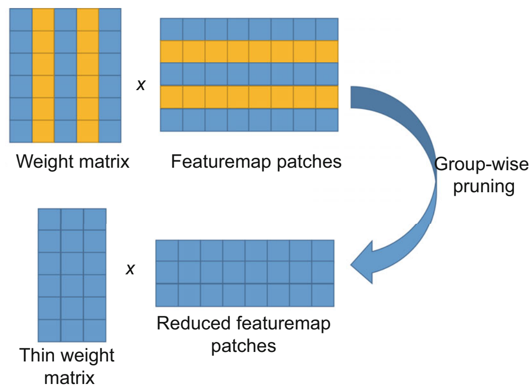

Fig. 2 Group-level pruning

3.4 Filter-level pruning

Filter-level pruning methods prune the convolutional filters or channels which make the deep networks thinner. After the filter pruning for one layer, the number of input channels of the next layer is also reduced. Thus, filter-level pruning is more efficient for accelerating 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 the current layer. By minimizing the feature map's reconstruction errors, they selected 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 $\boldsymbol{W}_i$, and then added sparse constraints on β_i. Then the channel selection problem can be casted into a least absolute shrinkage and selection operator (LASSO) regression problem. To minimize the feature map errors, they iteratively updated β and $\boldsymbol{W}$. Moreover, their method achieved a five-fold speed-up on VGG-16 network with little drop in accuracy. Instead of using additional selection weight β, Liu et al. (2017) proposed to leverage the scaling factor of the batch normalization layer to evaluate the importance of the filters. By pruning the channels with near-zero scaling factors, they could prune filters without introducing overhead into the networks.

4 Low-rank approximation

The convolutional kernel of a convolutional layer $\boldsymbol{W} \in \mathbb{R}^{w \cdot h \cdot c \cdot n}$ is a 4D tensor. These four dimensions correspond to the kernel width, kernel height, and the numbers of input and output channels, respectively. Note that by merging some of the dimensions, the 4D tensor can be transformed into a t-dimensional ($t = 1, 2, 3, 4$) tensor. The motivation behind low-rank decomposition is to find an approximate tensor $\hat{\boldsymbol{W}}$ that is close to $\boldsymbol{W}$ but facilitates a more efficient computation. Many low-rank based methods have been proposed by the community; two key differences are in how to rearrange the four dimensions, and on which dimension the low-rank constraint is imposed. Here we divide the low-rank based methods roughly into three categories, according to how many components the filters are decomposed into: two-, three-, and four-component decompositions.

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 $w \cdot h$ into $w \cdot 1$ and $1 \cdot h$ filters. They achieved a 4.5-fold speed-up for a CNN trained on a text character recognition dataset, with a 1% accuracy drop.

Singular value decomposition (SVD) is a popular low-rank matrix decomposition method. By merging dimensions w, h, and c, the kernel becomes a 2D matrix of size $(w \cdot h \cdot c) \cdot n$, on which the SVD

method can be conducted. Denil et al. (2013) used SVD to reduce the network redundancy. SVD was also investigated by Zhang X et al. (2015). The filters were replaced by two filter banks: one consisting of d filters of shape $w \cdot h \cdot c$ and the other composed of n filters of shape $1 \times 1 \cdot d$, where 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 the ImageNet classification task, this two-component SVD decomposition method achieved a three-fold theoretical speed-up at a cost of about 1.66% increased top-5 error.

Similarly, another SVD 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 $c \cdot (w \cdot h \cdot n)$. By selecting the rank to d, the convolution can be decomposed first by a $1 \times 1 \cdot c \cdot d$ convolution and then by a $w \cdot h \cdot d \cdot n$ convolution. These two decompositions are symmetric.

4.2 Three-component decomposition

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

If we use SVD along the input channel dimension for the first tensor after the two-component decomposition, we can obtain the Tucker decomposition format as proposed by Kim et al. (2015). These three components are convolutions of spatial sizes 1×1, $w \cdot h$, and 1×1. Note that instead of using the two-step SVD, Kim et al. (2015) used the Tucker decomposition method directly to obtain these three components. Their method achieved a 4.93-fold theoretical speed-up at a cost of 0.5% increased top-5 accuracy drop.

To further reduce 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 $w \cdot h$ convolution also requires 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 cannot be used. 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, BTD can be seen as a Tucker decomposition where the second decomposed tensor is a block diagonal tensor. By using this decomposition, they achieved a 7.4% actual speed-up for the VGG-16 model, at a cost of a 1.3% increased top-5 error. Their method also achieves a high speed-up for object detection and image retrieval tasks as reported in Wang et al. (2018).

4.3 Four-component decomposition

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

5 Network quantization

Quantization is an approach for many compression and acceleration applications. It has wide applications in image compression, information retrieval, etc. Many quantization methods have also been in-

Table 2 Comparison of fixed-point quantization methods according to which part is quantized and whether the training and testing stages 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-5 bits	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	2 bits	Full	No	Yes
DoReFa-Net (Zhou et al., 2016)	Binary	1–4 bits	6 bits, 8 bits, Full	Yes	Yes

BWN: binary weight network; BWNH: training binary weight networks via hashing; TWN: ternary weight network; FFN: fixed-point factorized network; INQ: incremental network quantization; BNN: binarized neural network; HWGQ: half-wave Gaussian quantization

vestigated for network acceleration and compression. We categorize these methods into two main groups: (1) scalar and vector quantization, which may need a codebook for quantization; (2) fixed-point quantization.

5.1 Scalar and vector quantization

Scalar and vector quantization techniques have a long history, and they were originally 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 the quantization centers. In general, the number of quantization centers is far smaller than that of original data. In addition, quantization codes can be encoded through a lossless encoding method (e.g., Huffman coding), or just represented as low-bit fixed points. Thus, scalar or vector quantization can achieve a 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, the product quantization (PQ) algorithm (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, the PQ algorithm can compress the fully connected layers with little loss. As Gong et al. (2014) compressed only the fully connected layers, Wu et al. (2016) and Cheng et al. (2017) proposed using the PQ algorithm to simultaneously accelerate and compress convolutional neural networks. They proposed quantizing the convolutional filters layer by layer by minimizing the feature map's reconstruction loss. During the inference phase, a look-up table was built by pre-computing the inner product between feature map patches and codebooks, and then the output feature map can be calculated by simply accessing the look-up table. By using this method, they could achieve a 4–6-fold speed-up and a 15–20-fold compression ratio with little accuracy loss.

5.2 Fixed-point quantization

Fixed-point quantization is an effective approach for lowering the resource consumption of a network. Based on which part is quantized, two main categories can be classified, i.e., weight and activation quantizations. There has been some other work that tried to also quantize gradients, resulting in acceleration at the network training stage. Here, we review mainly weight and activation quantization methods, which accelerate the test-phase computation. Table 2 summarizes these methods according to which part is quantized and whether the training and testing stages can be accelerated.

5.2.1 Fixed-point quantization of weights

Fixed-point weight quantization is a fairly mature topic in network acceleration and compression. Hammerstrom (2012) proposed a very large

scale integration (VLSI) architecture for network acceleration using 8-bit input and output, and 16-bit internal representation. Holi and Hwang (1993) provided a theoretical analysis of error caused by low-bit quantization to determine the bit-width for a multilayer perceptron. They showed that an 8–16-bit quantization was sufficient for training small neural networks. This early work focused on mainly simple multilayer perceptrons. A more recent work (Chen et al., 2014) showed that it is necessary to use 32-bit fixed-point numbers for the convergence of a convolutional neural network trained on the MNIST database. By using stochastic rounding, Gupta et al. (2015) found that it is sufficient to use 16-bit fixed-point numbers to train a convolutional neural network on the MNIST database. In addition, 8-bit fixed-point quantization was investigated by Dettmers (2015) to speed up the convergence of deep networks in parallel training. Logarithmic data representation was also investigated by Miyashita et al. (2016).

Recently, many lower-bit quantization or even binary and ternary quantization methods have been investigated. Cheng et al. (2015) introduced the expectation backpropagation (EBP), which uses the variational Bayes method to binarize the network. The BinaryConnect method proposed by Courbariaux et al. (2015) constrains all weights to be either $+1$ or -1. By training from scratch, the BinaryConnect can even outperform the floating-point counterpart on the CIFAR-10 image classification dataset (Krizhevsky and Hinton, 2009). Using binary quantization, the network can be compressed by a factor of about 32 compared with 32-bit floating-point networks. Most of the floating-point multiplication can also be eliminated (Lin et al., 2015). Rastegari et al. (2016) proposed the binary weight network (BWN), which was among the earliest work that achieved good results on the large ImageNet dataset (Russakovsky et al., 2015). Loss-aware binarization was proposed by Hou et al. (2016), which can directly minimize the classification loss with respect to the binarized weights. Hu et al. (2018) proposed a novel approach called 'BWNH' to train binary weight networks via hashing, which outperformed other weight binarization methods by a large margin. Ternary quantization was also used in Hwang and Sung (2014). Li et al. (2016) proposed the ternary weight network (TWN), which was similar to BWN but constrained all weights to be ternary values among $\{-1, 0, +1\}$. A TWN outperforms a BWN by a large margin on deep models like ResNet. Trained ternary quantization proposed by Zhu C et al. (2016) learns both ternary values and ternary assignments at the same time by using back-propagation. It achieves comparable results on the AlexNet model. Different from previous quantization methods, the incremental network quantization (INQ) method proposed by Zhou et al. (2017) gradually turns all weights into a logarithmic format in a multi-step manner. This incremental quantization strategy can lower the quantization error during each stage, and thus make the quantization problem much easier. All these low-bit quantization methods discussed above directly quantize the full-precision weight into a fixed-point format. Wang and Cheng (2017) proposed a different quantization strategy. In stead of direct quantization, they proposed using a fixed-point factorized network (FFN) to quantize all weights into ternary values. This fixed-point decomposition method can significantly lower the quantization error. The FFN method achieves comparable results on commonly used deep models such as AlexNet, VGG-16, and ResNet.

5.2.2 Fixed-point quantization of activations

Given only weight quantization, there is also a need for the time-consuming FLOPs. If the activations are also quantized into fixed-point values, the network can be executed efficiently by only fixed-point operations. Many activation quantization methods have also been proposed by the deep learning community. The bitwise neural network was proposed by Kim and Smaragdis (2016). A binarized neural network (BNN) is one of the first to quantize both weights and activations into either -1 or $+1$. A BNN achieves a comparable accuracy with the full-precision baseline on the CIFAR-10 dataset. To extend a BNN for the ImageNet classification task, Tang et al. (2017) improved the training strategies of BNN. A much higher accuracy was reported using these strategies. Based on a BWN, Rastegari et al. (2016) further quantized all activations into binary values, making the network into an XNOR-Net. Compared with a BNN, the XNOR-Net can achieve a much higher accuracy on the ImageNet dataset. To further understand the effect of bit-width on the training of DNNs, Zhou et al.

(2016) proposed DoReFa-Net by investigating the effects of different bit-widths for weights and activations as well as gradients. By using batch normalization, 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 the HWGQ method, with 2-bit activations and binary weights.

6 Teacher–student network

The teacher–student network is different from the network compression or acceleration methods since it trains a student network using a teacher network, and the student network can be designed with a different network architecture. Generally speaking, a teacher network is a large neural network or ensemble of neural networks, while a student network is a compact and efficient neural network. By using the dark knowledge transferred from the teacher network, the student network can achieve a higher accuracy than training merely through 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 the teacher network. Following this line of thinking, 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 their widths, a deeper student network would have a higher accuracy. Besides, Romero et al. (2014) used both intermediate layers' feature maps and soft outputs of the teacher network to train the student network. Rather than mimicking the intermediate layers' feature maps, 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 the layers' activations and their method can achieve a higher accuracy than FitNets.

7 Compact network design

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

Lin et al. (2013) proposed a network-in-network architecture, where a 1×1 convolution was used to increase the network capacity while keeping the overall computational complexity small. To reduce the storage requirement of the CNN models, they also proposed removing the fully connected layer and using a global average pooling. These strategies have also been used by many state-of-the-art CNN models like GoogLeNet (Szegedy et al., 2015) and ResNet (He et al., 2016).

Branching (multiple group convolution) is another commonly used strategy for lowering network complexity. It was explored in the work of GoogLeNet proposed by Szegedy et al. (2015). By largely using 1×1 convolution and the branching strategy, SqueezeNet proposed by Iandola et al. (2016) achieves about 50-fold compression over AlexNet, with a comparable accuracy. By branching, the work of ResNeXt proposed by Xie et al. (2017) can achieve a much higher accuracy than ResNet (He et al., 2016) at the same computational budget. The depth-wise convolution proposed in MobileNet by Howard et al. (2017) takes the branching strategy to the extreme; i.e., the number of branches equals the number of I/O channels. The resulting MobileNet can be 32-fold smaller and 27-fold faster than the VGG-16 model, with a comparable image classification accuracy on ImageNet. When using depth-wise convolution and 1×1 convolution as in MobileNet, most of the computation and parameters reside in the 1×1 convolutional layers. One strategy to further lower the complexity of the 1×1 convolution is to use multiple groups. ShuffleNet proposed by Zhang et al. (2017) introduces the channel shuffle operation to increase the information change within multiple groups, which can prominently increase the representational power of the networks. This method achieves about a 13-fold actual speedup over AlexNet with a comparable accuracy.

8 Hardware accelerator

8.1 Background

DNNs provide impressive performance for various tasks while suffering from degrees of computational complexity. Traditionally, algorithms based

on DNNs should be executed on general purpose platforms such as CPUs and GPUs; however, this works at the expense of unexpected power consumption and oversized resource utilization for both computation and storage. In recent years, there have been an increasing number of applications based on embedded systems, including autonomous vehicles, unmanned drones, security cameras, etc. Considering the demands for high performance, light weight, and low power consumption on these devices, CPU/GPU-based solutions are no longer suitable. In this scenario, FPGA/ASIC-based hardware accelerators are gaining a popularity as efficient alternatives.

8.2 General architecture

The deployment of a 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, it is usually carried out on GPUs off-line. During the inference phase, the pre-trained network parameters can be loaded either from the cloud or the dedicated off-chip memory. More recently, hardware accelerators for training have received a widespread attention (Ko et al., 2017; Venkataramani et al., 2017; Yang, 2017), but in this section we focus mainly on the inference phase in embedded settings.

Typically, an accelerator is composed of five parts: data buffers, parameter buffers, processing elements, global controller, and off-chip transfer manager (Fig. 3). The data buffers are used to cache input images, intermediate data, and output predictions, while the weight buffers are used mainly to cache convolutional filters. Processing elements are a collection of basic computing units that execute multiply-adds, non-linearity, and any other function such as normalization and quantization. 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 has been widely adopted in hardware acceleration. For computing-intensive operations such as multiply-adds, it is efficient to fit them on hardware for a high throughout; otherwise, data pre-processing, softmax, and any other graphic operation can be placed on CPU/GPU for low latency processing.

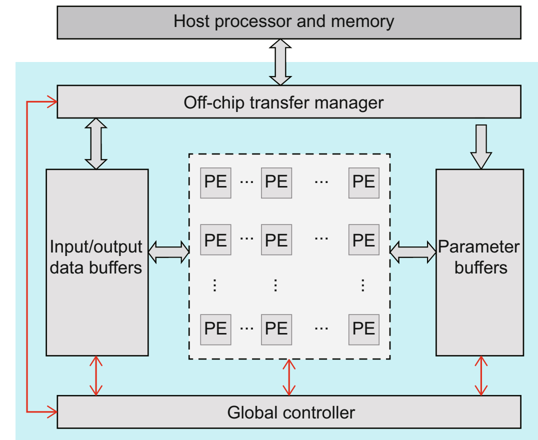

Fig. 3 General architecture of an accelerator on dedicated hardware (PE: processing element)

8.3 Processing elements

Among all of the accelerators, largest differences exist in the processing elements as they are designed for the majority of computing tasks in deep networks, such as massive multiply-add operations, normalization (batch normalization or local response normalization), and 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, and this architecture is designed mainly for fully investing in data reuse and parallelism. However, there are many accelerators that operate with only one processing element in consideration of lower data movement and resource conservation (Zhang C et al., 2015; Ma et al., 2017c).

8.4 Optimizing for a high throughput

Since the majority of the computations in a network are matrix–matrix/matrix–vector multiplication, it is critical to deal with the massive nested loops to achieve a high throughput. Loop optimization is one of the most frequently adopted techniques in accelerator design (Zhang C et al., 2015; Alwani et al., 2016; Suda et al., 2016; Ma et al., 2017b; Xiao et al., 2017; Li et al., 2018), including loop tiling, loop unrolling, loop interchange, etc. Loop tiling is used to divide all of the data into multiple small blocks to alleviate the pressure of on-chip storage (Alwani et al., 2016; Qiu et al., 2016; Ma et al., 2017b), while loop unrolling attempts to improve the parallelism of the computing engine for a high speed (Qiu et al., 2016; Ma et al., 2017b). Loop interchange determines the sequential

computation order of the nested loops because different computation orders can result in significant differences in performance. The well-known systolic array can be seen as a combination of the loop optimization methods listed above, which leverage the nature of data locality and weight sharing in the network to achieve a high throughput (Jouppi, 2017; Wei et al., 2017).

Single instruction multiple data (SIMD) based computation is another way for achieving a high throughput. Nguyen et al. (2017) presented a method to pack two low-bit multiplications into a single digital signal processing (DSP) block to double the computation, and Price et al. (2017) also proposed an SIMD-based architecture for speech recognition.

8.5 Optimizing for a low energy consumption

Existing work attempts to reduce the energy consumption of a hardware accelerator from both computing and I/O perspectives. Horowitz (2014) systematically illustrated the energy cost in terms of arithmetic operations and memory accesses. He demonstrated that operations based on integers are much more cheaper than their float-point counterparts, and lower-bit integers are better. Therefore, most existing accelerators adopt low-bit or even binary data representation (Nurvitadhi et al., 2017; Umuroglu et al., 2017; Zhao et al., 2017) to preserve energy efficiency. More recently, logarithmic computation that transfers multiplications into bit-shift operations has also shown its promise in energy saving (Lee et al., 2017; Gudovskiy and Rigazio, 2017; Tann et al., 2017).

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

Off-chip data transfers happen inordinately 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 the dynamic random-access memory (DRAM) access is several orders of magnitude of the static random-access memory (SRAM) access, and therefore reducing off-chip transfers is a critical issue. Shen et al. (2017) designed a flexible data buffing scheme to reduce bandwidth requirements, and Alwani et al. (2016) and Xiao et al. (2017) proposed a 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 a 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 Y et al. (2017) presented an energy-efficient row stationary scheme to reduce data movements, and Zhu J et al. (2016) attempted to reduce power consumption via low-rank approximation.

8.6 Design automation

Recently, design automation frameworks that automatically map DNNs onto hardware are receiving a wider attention. Wang et al. (2016), Sharma et al. (2016), Venieris and Bouganis (2016), and Wei et al. (2017) proposed frameworks that automatically generate synthesizable accelerator for a given network. Ma et al. (2017a) presented a register transfer level (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 have been many new techniques from both the algorithm side and circuit side that have been adopted to implement fast and energy-efficient accelerators. Stochastic computing representing continuous values through streams of random bits has been investigated for hardware acceleration of DNNs (Kim K et al., 2016; Ren et al., 2017; Sim and Lee, 2017). On the hardware side, resistive random-access memory (RRAM) based accelerators (Xia et al., 2016; Chen L et al., 2017) and the use of 3D DRAM (Kim D et al., 2016; Gao et al., 2017) have received a greater attention.

9 Future trends and discussion

In this section, we discuss some possible future directions in this field, i.e., non-fine-tuning or unsupervised compression. Most of the existing methods, including network pruning, low-rank compression, and quantization, need labeled data to retrain the network for accuracy retention. The problems are two-fold. First, labeled data is sometimes unavailable, as in medical images. Another problem is that retraining requires considerable human efforts as well as professional knowledge. These two problems raise the need for unsupervised compression or even fine-tuning-free compression methods.

Scalable (self-adaptive) compression: Current compression methods have many hyperparameters that need to be determined ahead of time, e.g., the sparsity of the network pruning, the rank of the decomposition-based methods, or the bit-width of fixed-point quantization methods. The selection of these hyperparameters is a tedious work, which also requires professional experience. Thus, the investigation of methods that do not rely on human-designed hyperparameters is a promising research topic. One direction may be to use annealing methods, or reinforcement learning.

Network acceleration for object detection: Most of the model acceleration methods are optimized for image classification, yet very little effort has been devoted to the acceleration of other computer vision tasks such as object detection. It seems that model acceleration methods for image classification can be used directly for detection. However, DNNs 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 greater accuracy drops than with image classification. One reason for this phenomenon may be that object detection requires more complex feature representation than image classification. The design of model acceleration methods for object detection represents a challenge.

Hardware-software co-design: To accelerate the 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 makes it difficult to put this into practice. Recent advances in deep learning algorithms and hardware accelerators demonstrate that it is highly desirable to design hardware-efficient algorithms according to the low-level features of specific hardware platforms. This co-design methodology will be a trend in future work.

10 Conclusions

DNNs have provided impressive performance while suffering from a huge computational complexity and a high energy expenditure. In this paper, we have provided a survey of recent advances in efficient processing of DNNs from both the algorithm and hardware points of view. In addition, we pointed out a few topics that deserve further investigations in the future.

References

Albericio J, Judd P, Hetherington T, et al., 2016. Cnvlutin: ineffectual-neuron-free deep neural network computing. Proc 43rd Int Symp on Computer Architecture, p.1-13. https://doi.org/10.1145/3007787.3001138

Alwani M, Chen H, Ferdman M, et al., 2016. Fused-layer CNN accelerators. 49th Annual IEEE/ACM Int Symp on MICRO, p.1-12. https://doi.org/10.1109/MICRO.2016.7783725

Anwar S, Hwang K, Sung W, 2017. Structured pruning of deep convolutional neural networks. *ACM J Emerg Technol Comput Syst*, 13(3), Article 32. https://doi.org/10.1145/3005348

Cai Z, He X, Sun J, et al., 2017. Deep learning with low precision by half-wave Gaussian quantization. IEEE Computer Society Conf on Computer Vision and Pattern Recognition, p.5918-5926.

Chen L, Li J, Chen Y, et al., 2017. Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar. Proc Conf on Design, Automation and Test in Europe Conf and Exhibition, p.19-24.

Chen Y, Sun N, Temam O, et al., 2014. DaDianNao: a machine-learning supercomputer. Proc 47th Annual IEEE/ACM Int Symp on Microarchitecture, p.609-622. https://doi.org/10.1109/MICRO.2014.58

Chen Y, Krishna T, Emer J, et al., 2017. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J Sol-Stat Circ*, 52(1): 127-138. https://doi.org/10.1109/JSSC.2016.2616357

Cheng J, Wu J, Leng C, et al., 2017. Quantized CNN: a unified approach to accelerate and compress convolutional networks. *IEEE Trans Neur Netw Learn Syst*, 99:1-14. https://doi.org/10.1109/TNNLS.2017.2774288

Cheng Z, Soudry D, Mao Z, et al., 2015. Training binary multilayer neural networks for image classification using expectation backpropagation. http://arxiv.org/abs/1503.03562

Courbariaux M, Bengio Y, David J, 2015. Binaryconnect: training deep neural networks with binary weights during propagations. NIPS, p.3123-3131.

Denil M, Shakibi B, Dinh L, et al., 2013. Predicting parameters in deep learning. NIPS, p.2148-2156.

Dettmers T, 2015. 8-bit approximations for parallelism in deep learning. http://arxiv.org/abs/1511.04561

Gao M, Pu J, Yang X, et al., 2017. TETRIS: scalable and efficient neural network acceleration with 3D memory. Proc 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems, p.751-764. https://doi.org/10.1145/3093337.3037702

Gong Y, Liu L, Yang M, et al., 2014. Compressing deep convolutional networks using vector quantization. http://arxiv.org/abs/1412.6115

Gudovskiy D, Rigazio L, 2017. ShiftCNN: generalized low-precision architecture for inference of convolutional neural networks. http://arxiv.org/abs/1706.02393

Guo Y, Yao A, Chen Y, 2016. Dynamic network surgery for efficient DNNs. NIPS, p.1379-1387.

Gupta S, Agrawal A, Gopalakrishnan K, et al., 2015. Deep learning with limited numerical precision. Proc 32nd Int Conf on Machine Learning, p.1737-1746.

Hammerstrom D, 2012. A VLSI architecture for high-performance, low-cost, on-chip learning. IJCNN Int Joint Conf on Neural Networks, p.537-544. https://doi.org/10.1109/IJCNN.1990.137621

Han S, Mao H, Dally W, 2015a. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. http://arxiv.org/abs/1510.00149

Han S, Pool J, Tran J, et al., 2015b. Learning both weights and connections for efficient neural network. NIPS, p.1135-1143.

Han S, Liu X, Mao H, et al., 2016. EIE: efficient inference engine on compressed deep neural network. ACM/IEEE 43rd Annual Int Symp on Computer Architecture, p.243-254. https://doi.org/10.1109/ISCA.2016.30

Han S, Kang J, Mao H, et al., 2017. ESE: efficient speech recognition engine with sparse LSTM on FPGA. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.75-84. https://doi.org/10.1145/3020078.3021745

Hassibi B, Stork D, 1993. Second order derivatives for network pruning: optimal brain surgeon. NIPS, p.164-171.

He K, Zhang X, Ren S, et al., 2016. Deep residual learning for image recognition. Proc IEEE Conf on Computer Vision and Pattern Recognition, p.770-778.

He Y, Zhang X, Sun J, 2017. Channel pruning for accelerating very deep neural networks. http://arxiv.org/abs/1707.06168

Hinton G, Vinyals O, Dean J, 2015. Distilling the knowledge in a neural network. http://arxiv.org/abs/1503.02531

Holi J, Hwang J, 1993. Finite precision error analysis of neural network hardware implementations. *IEEE Trans Comput*, 42(3):281-290. https://doi.org/10.1109/12.210171

Horowitz M, 2014. 1.1 computing's energy problem (and what we can do about it). IEEE Int Solid-State Circuits Conf Digest of Technical Papers, p.10-14. https://doi.org/10.1109/ISSCC.2014.6757323

Hou L, Yao Q, Kwok J, 2016. Loss-aware binarization of deep networks. http://arxiv.org/abs/1611.01600

Howard A, Zhu M, Chen B, et al., 2017. Mobilenets: efficient convolutional neural networks for mobile vision applications. http://arxiv.org/abs/1704.04861

Hu Q, Wang P, Cheng J, 2018. From hashing to CNNs: training binary weight networks via hashing. 32nd AAAI Conf on Artificial Intelligence, in press.

Hwang K, Sung W, 2014. Fixed-point feedforward deep neural network design using weights $+1$, 0, and -1. IEEE Workshop on Signal Processing Systems, p.1-6. https://doi.org/10.1109/SiPS.2014.6986082

Iandola F, Han S, Moskewicz M, et al., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. http://arxiv.org/abs/1602.07360

Jaderberg M, Vedaldi A, Zisserman A, 2014. Speeding up convolutional neural networks with low rank expansions. http://arxiv.org/abs/1405.3866

Jegou H, Douze M, Schmid C, 2011. Product quantization for nearest neighbor search. *IEEE Trans Patt Anal Mach Intell*, 33(1):117-128. https://doi.org/10.1109/TPAMI.2010.57

Jouppi N, 2017. In-datacenter performance analysis of a tensor processing unit. Proc 44th Annual Int Symp on Computer Architecture, p.1-12. https://doi.org/10.1145/3140659.3080246

Kim D, Kung J, Chai S, et al., 2016. Neurocube: a programmable digital neuromorphic architecture with high-density 3D memory. ACM/IEEE 43rd Annual Int Symp on Computer Architecture, p.380-392. https://doi.org/10.1109/ISCA.2016.41

Kim K, Kim J, Yu J, et al., 2016. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. Proc 53rd Annual Design Automation Conf, Article 124. https://doi.org/10.1145/2897937.2898011

Kim M, Smaragdis P, 2016. Bitwise neural networks. http://arxiv.org/abs/1601.06071

Kim YD, Park E, Yoo S, et al., 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. http://arxiv.org/abs/1511.06530

Ko J, Mudassar B, Na T, et al., 2017. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/3061639.3062228

Krizhevsky A, Hinton G, 2009. Learning Multiple Layers of Features from Tiny Images. MS Thesis, Department of Computer Science, University of Toronto, Toronto, Canada.

Krizhevsky A, Sutskever I, Hinton G, 2012. Imagenet classification with deep convolutional neural networks. NIPS, p.1097-1105.

Lebedev V, Lempitsky V, 2016. Fast ConvNets using group-wise brain damage. IEEE Conf 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. http://arxiv.org/abs/1412.6553

LeCun Y, Denker J, Solla S, et al., 1989. Optimal brain damage. NIPS, p.598-605.

Lee EH, Miyashita D, Chai E, et al., 2017. LogNet: energy-efficient neural networks using logarithmic computation. IEEE Int Conf on Acoustics, Speech and Signal Processing, p.5900-5904.
https://doi.org/10.1109/ICASSP.2017.7953288

Li F, Zhang B, Liu B, 2016. Ternary weight networks.
http://arxiv.org/abs/1605.04711

Li G, Li F, Zhao T, et al., 2018. Block convolution: towards memory-efficient inference of large-scale CNNs on FPGA. Design Automation and Test in Europe, in press.

Lin M, Chen Q, Yan S, 2013. Network in network.
http://arxiv.org/abs/1312.4400

Lin Z, Courbariaux M, Memisevic R, et al., 2015. Neural networks with few multiplications.
http://arxiv.org/abs/1510.03009

Liu S, Du Z, Tao J, et al., 2016. Cambricon: an instruction set architecture for neural networks. Proc 43rd Int Symp on Computer Architecture, p.393-405.
https://doi.org/10.1145/3007787.3001179

Liu Z, Li J, Shen Z, et al., 2017. Learning efficient convolutional networks through network slimming. IEEE Int Conf on Computer Vision, p.2736-2744

Luo J, Wu J, Lin W, 2017. ThiNet: a filter level pruning method for deep neural network compression.
http://arxiv.org/abs/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. 27th Int Conf on Field Programmable Logic and Applications, p.1-8. https://doi.org/10.23919/FPL.2017.8056824

Ma Y, Cao Y, Vrudhula S, et al., 2017b. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.45-54.
https://doi.org/10.1145/3020078.3021736

Ma Y, Kim M, Cao Y, et al., 2017c. End-to-end scalable FPGA accelerator for deep residual networks. IEEE Int Symp on Circuits and Systems, p.1-4.
https://doi.org/10.1109/ISCAS.2017.8050344

Mao H, Han S, Pool J, et al., 2017. Exploring the regularity of sparse structure in convolutional neural networks.
http://arxiv.org/abs/1705.08922

Miyashita D, Lee E, Murmann B, 2016. Convolutional neural networks using logarithmic data representation.
http://arxiv.org/abs/1603.01025

Nguyen D, Kim D, Lee J, 2017. Double MAC: doubling the performance of convolutional neural networks on modern FPGAs. Design, Automation & Test in Europe Conf & Exhibition, p.890-893.
https://doi.org/10.23919/DATE.2017.7927113

Nurvitadhi E, Hillsboro, Venkatesh G, et al., 2017. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.5-14.
https://doi.org/10.1145/3020078.3021740

Parashar A, Rhu M, Mukkara A, et al., 2017. SCNN: an accelerator for compressed-sparse convolutional neural networks. Proc 44th Annual Int Symp on Computer Architecture, p.27-40.
https://doi.org/10.1145/3140659.3080254

Price M, Glass J, Chandrakasan A, 2017. 14.4 a scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating. IEEE Int Solid-State Circuits Conf, p.244-245.
https://doi.org/10.1109/ISSCC.2017.7870352

Qiu JT, Wang J, Yao S, et al., 2016. Going deeper with embedded FPGA platform for convolutional neural network. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.26-35.
https://doi.org/10.1145/2847263.2847265

Rastegari M, Ordonez V, Redmon J, et al., 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. European Conf on Computer Vision, p.525-542.
https://doi.org/10.1007/978-3-319-46493-0_32

Ren A, Li Z, Ding C, et al., 2017. SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing. Proc 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems, p.405-418. https://doi.org/10.1145/3093336.3037746

Romero A, Ballas N, Kahou S, et al., 2014. FitNets: hints for thin deep nets. http://arxiv.org/abs/1412.6550

Russakovsky O, Deng J, Su H, et al., 2015. Imagenet large scale visual recognition challenge. *Int J Comput Vis*, 115(3):211-252.
https://doi.org/10.1007/s11263-015-0816-y4

Sharma H, Park J, Mahajan D, et al., 2016. From high-level deep neural models to FPGAs. 49th Annual IEEE/ACM Int Symp on Microarchitecture, p.1-21.
https://doi.org/10.1109/MICRO.2016.7783720

Shen Y, Ferdman M, Milder P, 2017. Escher: a CNN accelerator with flexible buffering to minimize off-chip transfer. IEEE 25th Annual Int Symp on Field-Programmable Custom Computing Machines, p.93-100.
https://doi.org/10.1109/FCCM.2017.47

Sim H, Lee J, 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. Proc 54th Annual Design Automation Conf, Article 29.
https://doi.org/10.1145/3061639.3062290

Simonyan K, Zisserman A, 2014. Very deep convolutional networks for large-scale image recognition.
http://arxiv.org/abs/1409.1556

Suda N, Chandra V, Dasika G, et al., 2016. Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.16-25. https://doi.org/10.1145/2847263.2847276

Szegedy C, Liu W, Jia Y, et al., 2015. Going deeper with convolutions. Conf on Computer Vision and Pattern Recognition, p.1-9.
https://doi.org/10.1109/CVPR.2015.7298594

Tang W, Hua G, Wang L, 2017. How to train a compact binary neural network with high accuracy? 31st AAAI Conf on Artificial Intelligence, p.2625-2631.

Tann H, Hashemi S, Bahar I, et al., 2017. Hardware-software codesign of accurate, multiplier-free deep neural networks. 54th ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/3061639.3062259

Umuroglu Y, Fraster N, Gambardella G, et al., 2017. FINN: a framework for fast, scalable binarized neural network inference. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.65-74.
https://doi.org/10.1145/3020078.3021744

Venieris S, Bouganis C, 2016. fpgaAConvNet: a framework for mapping convolutional neural networks on FPGAs. IEEE 24th Annual Int Symp on Field-Programmable Custom Computing Machines, p.40-47. https://doi.org/10.1109/FCCM.2016.22

Venkataramani S, Ranjan A, Banerjee S, et al., 2017. ScaleDeep: a scalable compute architecture for learning and evaluating deep networks. Proc 44th Annual Int Symp on Computer Architecture, p.13-26. https://doi.org/10.1145/3079856.3080244

Wang P, Cheng J, 2016. Accelerating convolutional neural networks for mobile applications. Proc ACM on Multimedia Conf, p.541-545. https://doi.org/10.1145/2964284.2967280

Wang P, Cheng J, 2017. Fixed-point factorized networks. IEEE Conf on Computer Vision and Pattern Recognition, p.4012-4020.

Wang P, Hu Q, Fang Z, et al., 2018. Deepsearch: a fast image search framework for mobile devices. *ACM Trans Multim Comput Commun Appl*, 14(1), Article 6. https://doi.org/10.1145/3152127.

Wang Y, Xu J, Han Y, et al., 2016. Deepburning: automatic generation of FPGA-based learning accelerators for the neural network family. Proc 53rd Annual Design Automation Conf, Article 110. https://doi.org/10.1145/2897937.2898003

Wei SC, Yu CH, Zhang P, et al., 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. 54th ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/3061639.3062207

Wen W, Wu C, Wang Y, et al., 2016. Learning structured sparsity in deep neural networks. NIPS, p.2074-2082.

Wu J, Leng C, Wang Y, et al., 2016. Quantized convolutional neural networks for mobile devices. Proc IEEE Conf on Computer Vision and Pattern Recognition, p.4820-4828.

Xia L, Tang T, Huangfu W, et al., 2016. Switched by input: power efficient structure for RRAM-based convolutional neural network. 53rd ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/2897937.2898101

Xiao QC, Liang Y, Lu LQ, et al., 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. 54th ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/3061639.3062244

Xie S, Girshick R, Dollar P, et al., 2017. Aggregated residual transformations for deep neural networks. IEEE Conf on Computer Vision and Pattern Recognition, p.5987-5995. https://doi.org/10.1109/CVPR.2017.634

Yang H, 2017. TIME: a training-in-memory architecture for memristor-based deep neural networks. 54th ACM/EDAC/IEEE Design Automation Conf, p.1-6. https://doi.org/10.1145/3061639.3062326

Zagoruyko S, Komodakis N, 2016. Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. http://arxiv.org/abs/1612.03928

Zhang C, Li P, Sun GY, et al., 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.161-170. https://doi.org/10.1145/2684746.2689060

Zhang C, Fang Z, Pan P, et al., 2016a. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. IEEE/ACM Int Conf on Computer-Aided Design, p.1-8. https://doi.org/10.1145/2966986.2967011

Zhang C, Wu D, Sun J, et al., 2016b. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. Proc Int Symp on Low Power Electronics and Design, p.326-331. https://doi.org/10.1145/2934583.2934644

Zhang S, Du Z, Zhang L, et al., 2016. Cambricon-X: an accelerator for sparse neural networks. 49th Annual IEEE/ACM Int Symp on Microarchitecture, p.1-12. https://doi.org/10.1109/MICRO.2016.7783723

Zhang X, Zou J, He K, et al., 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans Patt Anal Mach Intell*, 38(10):1943-1955. https://doi.org/10.1109/TPAMI.2015.2502579

Zhang X, Zhou X, Lin M, et al., 2017. ShuffleNet: an extremely efficient convolutional neural network for mobile devices. http://arxiv.org/abs/1707.01083

Zhao R, Song WN, Zhang WT, et al., 2017. Accelerating binarized convolutional neural networks with software-programmable FPGAs. Proc ACM/SIGDA Int Symp on Field-Programmable Gate Arrays, p.15-24. https://doi.org/10.1145/3020078.3021741

Zhou A, Yao A, Guo Y, et al., 2017. Incremental network quantization: towards lossless CNNs with low-precision weights. http://arxiv.org/abs/1702.03044

Zhou S, Wu Y, Ni Z, et al., 2016. DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. http://arxiv.org/abs/1606.06160

Zhu C, Han S, Mao H, et al., 2016. Trained ternary quantization. http://arxiv.org/abs/1612.01064

Zhu J, Qian Z, Tsui C, 2016. LRADNN: high-throughput and energy-efficient deep neural network accelerator using low rank approximation. 21st Asia and South Pacific Design Automation Conf, p.581-586. https://doi.org/10.1109/ASPDAC.2016.7428074