Recent Advances in Convolutional Neural Network Acceleration

Qianru Zhang[a], Meng Zhang[a,*], Tinghuan Chen[b], Zhifei Sun[a], Yuzhe Ma[b], Bei Yu[b]

[a]*National ASIC System Engineering Technology Research Center, Southeast University, Nanjing, China*
[b]*Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong*

Abstract

In recent years, convolutional neural networks (CNNs) have shown great performance in various fields such as image classification, pattern recognition, and multi-media compression. Two of the feature properties, local connectivity and weight sharing, can reduce the number of parameters and increase processing speed during training and inference. However, as the dimension of data becomes higher and the CNN architecture becomes more complicated, the end-to-end approach or the combined manner of CNN is computationally intensive, which becomes limitation to CNN's further implementation. Therefore, it is necessary and urgent to implement CNN in a faster way. In this paper, we first summarize the acceleration methods that contribute to but not limited to CNN by reviewing a broad variety of research papers. We propose a taxonomy in terms of three levels, i.e. structure level, algorithm level, and implementation level, for acceleration methods. We also analyze the acceleration methods in terms of CNN architecture compression, algorithm optimization, and hardware-based improvement. At last, we give a discussion on different perspectives of these acceleration and optimization methods within each level. The discussion shows that the methods in each level still have large exploration space. By incorporating such a wide range of disciplines, we expect to provide a comprehensive reference for researchers who are interested in CNN acceleration.

Keywords: Convolutional neural network, Model compression, Algorithm optimization, Hardware acceleration

1. Introduction

Convolutional neural network (CNN) architectures have been around for over two decades. Compared with other neural network models such as multiple layer perceptron (MLP), CNN is designed to take multiple arrays as input and then process the input using convolution operator within a local field by mimicking eyes perceiving images. Therefore, it shows excellent performance in solving computer vision problems such as image classification, recognition and understanding [1, 2, 3]. It is also effective for a wide range of fields such as speech recognition that requires correlated speech spectral representations [4], VLSI physical design [5], multi-media compression [6] comparing with the traditional DCT transformation and compressive sensing methods [7, 8], and cancer detection from a series of condition changing images [9]. Moreover, many top players have been in a fever to play Go match with alphaGo recently, which has CNN implemented.

However, in order to receive good performance of prediction and accomplish more difficult goals, CNN architecture becomes deeper and more complicated. At the same time, more pixels are packed into one image thanks to high resolution acquisition devices. As a result, CNN training and inference are very computationally expensive and become limited for implementation due to its slow speed. Although acceleration and optimization for CNN have been explored since it was brought up, recently this seems to be keener as it has such good industrial impact.

Some companies have unveiled accelerators for deep learning inference that can be extensively used for CNN. Google's second generation Tensor Processing Unit (TPU) is designed for TensorFlow framework that has the increased performance of 92TFLOPS in peak and on-chip memory of 28MiB. It not only supports integers but also floating point calculations, which makes it more powerful in deep learning training [10]. NVIDIA launches an open source project called NVIDIA Deep Learning Accelerator (NVDLA) along with an open license that is ready for people who are interested in data intensive automotive products. It in-

[*]Corresponding author
Email address: zmeng@seu.edu.cn (Meng Zhang)

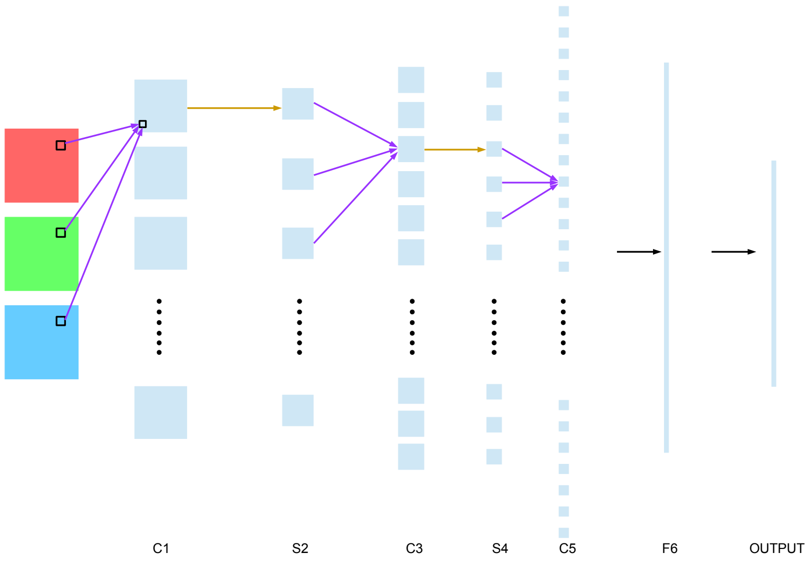


Figure 1: Illustration of LeNet-5.

cludes Verilog and C-model for the chip, Linux drivers, test suites, and kernel and user based software with development tools [11]. Intel's Nervana Neural Network Processor (NNP) has been announced recently for dealing with neural network matrix multiplications and convolutions. The memory architecture is designed for efficient operations and data movements without cache hierarchy, which improves the training performance of deep learning [12].

In this paper, we review many recent works and summarize acceleration methods not only in structure level and algorithm level, but also in implementation level. This paper differs from other deep neural network review papers in three aspects. **1) Topic:** Some review papers summarize the relevant work regarding deep neural networks in different applications such as computer vision, natural language processing and acoustic signal processing, among which CNN is only a component [13, 14, 15, 16, 17, 18]. They make systematic introduction for various kinds of neural networks that are fit for specific applications and provide a guide for people who want to implement deep neural networks in their fields. However, few of them mention acceleration methods, while our paper focuses on CNN and its acceleration methods. **2) Time:** Recent deep learning review papers are mostly historical [19, 20, 21]. They usually trace back the origins through over fifteen years to form a big picture of the neural network development, which is very inspiring to think over the origins. Our paper focuses on researches recently when hardware becomes limited and efficiency becomes the priority. **3) Taxonomy:** There are no reviews that incorporate hardware into algorithms since they are different disciplines. In this paper, we talk about the acceleration methods in three levels, because they are interwoven and highly dependent.

This survey paper is organized as following. In Section 2, an overview of modern CNN structure is given with different typical layers that the improvement is focused on. In Section 3 we present our taxonomy for recent CNN acceleration methods followed by the overview in three categories, including CNN compression in Section 4, algorithm optimization in Section 5, and hardware-oriented acceleration in Section 6. After that, in Section 7 a discussion is given on these methods from different perspectives. Finally, Section 8 concludes this paper with some future challenges.

2. Convolutional Neural Network

The modern convolutional neural networks proposed by LeCun [22] is a 7-layer (excluding the input layer) LeNet-5 structure. It has the following structure C1, S2, C3, S4, C5, F6, OUTPUT as shown in Figure 1, where C indicates convolutional layer, S indicates subsampling layer, and F indicates fully-connected layer. There are many modifications regarding the structure of CNNs in order to handle more complicated datasets and problems, such as AlexNet (8 layers) [23], GoogLeNet (22 layers) [27], VGG-16 (16 layers) [25], and ResNet (152 layers) [26]. Table 1 summarizes the state-of-the-art CNNs. In this table, Feature column summarizes the

Model	Layer Size	Configuration	Feature	Parameter Size	Application
LeNet [22]	7 layers	3C-2S-1F-RBF output layer		60,000	Document recognition
AlexNet [23]	8 layers	5C-3S-3F	Local response normalization	60,000,000	Image classification
NIN [24]	-	3mlpconv-global average pooling (S can be added in between the mlpconv)	mlpconv layer: 1C-3MLP; global average pooling	-	Image classification
VGG [25]	11-19 layers	VGG-16: 13C-5S-3F	Increased depth with stacked 3×3 kernels	133,000,000 to 144,000,000	Image classification and localization
ResNet [26]	Can be very deep (152 layers)	ResNet-152: 151C-2S-1F	Residual module	ResNet-20: 270,000; ResNet-1202: 19,400,000	Image classification, object detection
GoogLeNet [27]	22 layers	3C-9Inception-5S-1F	Inception module	6,797,700	Image classification, object detection
Xception [28]	37 layers	36C-5S-1F	Depth-wise separable convolutions	22,855,952	Image classification

Table 1: CNN model summary.
C: convolutional layer, S: subsampling layer, F: fully-connected layer

most important parts in each model [29, 30, 31]. Application column provides the fields that the methods were proposed for the first time. Fully-connected layer is followed by the Softmax layer except for LeNet and NIN. As we can see from the table, the number of parameters in modern CNNs is large, which usually takes a long time for training and for inference. Plus, higher dimensional input, large number of parameters, and complex CNN configuration challenge hardware in terms of processing element efficiency, memory bandwidth, off-chip memory, communication and so on.

Among these different structures, they share four key features including weight sharing, local connection, pooling, and the use of many layers [20]. There are some commonly used layers such as convolutional layers, subsampling layers (pooling layers), and fully-connected layers. Usually, there is a convolutional layer after the input. The convolutional layer is often followed by a subsampling layer. This combination repeats several times to increase the depth of CNN. The fully-connected layers are designed as the last few layers in order to map from extracted features to labels. These four layers are introduced as follows.

a) Input Layer: In CNNs, input layers usually take multiple arrays and are often size-fixed. Comparing to ordinary fully-connected neural networks, the CNN input do not need size-normalization and centralization, because CNN enjoys the characteristic of translation invariance [32].

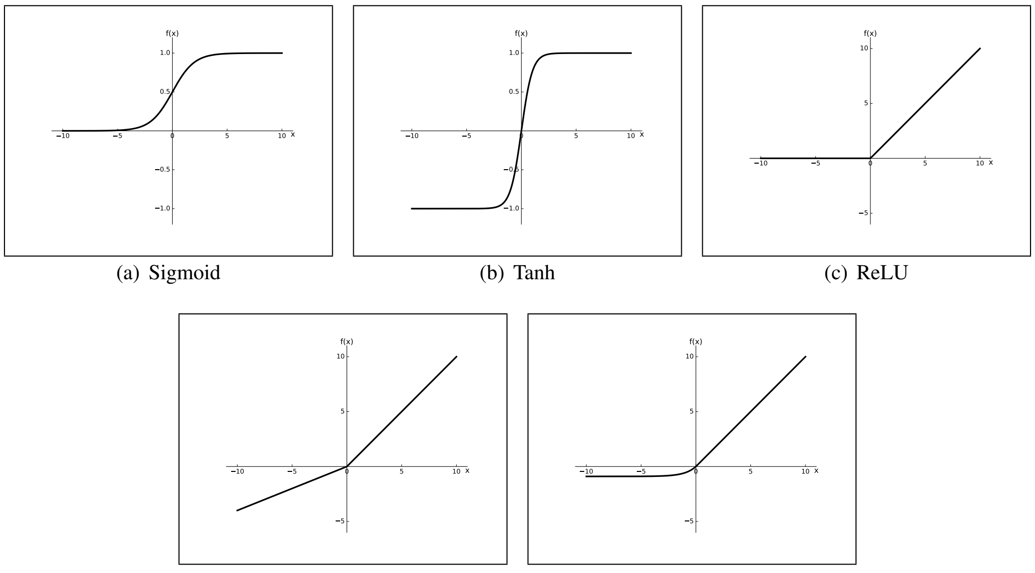
b) Convolutional Layer: As a key feature layer that makes CNNs different from other ordinary neural networks, neuron units of convolutional layers are first computed by convolution operation over small local patches of input, and then followed by activation functions (tanh, sigmoid, ReLU, etc.), and form a 2D feature map (3D feature map channel). In general, we have that

$$\mathbf{Z}_j = \sum_i \mathbf{X}_i * \mathbf{K}_{ij} + \mathbf{B}_j, \tag{1}$$

$$\mathbf{A}_j = f(\mathbf{Z}_j), \tag{2}$$

where $\mathbf{Z}_j$ represents the output from the convolution operation, $\mathbf{X}_i$ denotes the input to the convolutional layer, $\mathbf{K}_{ij}$ is the convolution kernel, and $\mathbf{B}_j$ is the additive bias. In the following equation, $\mathbf{A}_j$ is the output feature map of the convolutional layer and $f(\cdot)$ is an activation function.

Activation functions are mathematical operations over the input, which introduces non-linearity into neural networks and help catch non-linear features of the


(a) Sigmoid (b) Tanh (c) ReLU

(d) LeakyReLU / PReLU / RReLU (e) ELU

Figure 2: Activation function plot.

Function	Saturation	Definition	Parameter α	Plot
Sigmoid	Saturated	$f(x) = 1/(1+e^{-x})$	-	(a)
Tanh	Saturated	$f(x) = 2/(1+e^{-2x}) - 1$	-	(b)
ReLU	Non-saturated	$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$	-	(c)
LeakyReLU	Non-saturated	$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$	$\alpha \in (0,1)$	(d)
PReLU	Non-saturated	$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$	α is a learned parameter	(d)
RReLU	Non-saturated	$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$	$\alpha \sim$ uniform(a, b)	(d)
ELU	Non-saturated	$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$	α is a predefined parameter	(e)

Table 2: Activation function summary.

input data. There are various types of activation functions as summarized in Table 2 and Figure 2.

Sigmoid and Tanh are called saturated functions. As we can see from their definitions or plots, when the input is very small or very large, the output saturates at 0 or 1 for Sigmoid and -1 or 1 for Tanh. There are two problems with saturation. The gradients at saturated regions are almost zero, which dramatically decreases neurons backpropagation and makes it difficult to converge in the training phase. Furthermore, more attention needs to be paid in weight initialization when using saturated activation functions for the neural networks may not learn in the first place. To alleviate saturation problem, many non-saturated activations are proposed such as Rectified Linear Unit (ReLU) [33], Leaky ReLU [34], Parametric ReLU (PReLU) [35], Randomized Leaky ReLU (RReLU) [36], and Exponential Linear Unit (ELU) [37].

Convolution plays a very important role in CNN. On one hand, by weight sharing, neurons in the same feature map share the same parameters, which reduces dramatically the total number of parameters. In different spatial location, input may have some same features such as edges, points, angles, etc. Weight sharing makes the CNN less sensitive to location and shifting. On the other hand, since each convolution operation is targeted for a small patch of input, the extracted features remain intrinsic topology of the input that helps recognize patterns.

c) Subsampling Layer (pooling layer): Convolutional layers are usually followed by subsampling layers to reduce the feature map resolution. The amount of parameters and computation are also reduced accordingly. More formally,

$$\mathbf{Z}_j = \mathrm{down}(\mathbf{X}_j), \tag{3}$$

where down(·) represents a subsampling method.

Maximum operation and average operation are two typical subsampling methods and have been implemented in CNNs. In spite of max pooling and average pooling, some methods that work better in mitigating overfitting problems in CNN are proposed such as Lp pooling [38], stochastic pooling [39], and mixed pooling [40]. He *et al.* propose a pooling method called spatial pyramids pooling (SPP) that can output a fixed-length feature map and therefore can deal with various input image sizes [41]. Spectral pooling is a pooling method to reduce dimensionality in frequency, which preserves more information than spacial domain, and can be implemented in Fast Fourier Transform (FFT) based CNNs [42]. While multi-scale orderless pooling proposed by Gong *et al.* outperforms other methods in highly variable scene matching [43].

Different from convolution kernels, subsampling kernels are often hand-picked and remain unchanged during training and inference. There are two main reasons for subsampling. One is that by maximizing or averaging over the previous feature map, the size of feature map reduces. The other one is that by subsampling, the output feature map is more robust to distortions and errors of individual neuron units [44].

d) Fully-connected Layer: After several layers, high-level features are extracted and require mapping to labels. In fully-connected layer, neuron units are transformed from 2D into 1D. Each unit in the current layer is connected to all the units in the previous layer such like regular neural networks. It not only extracts features in a more complex way in order to dig deep for more information, but patterns in different locations are connected as well.

e) Output Layer: As a feed-forward neural network, the output layer neuron units are fixed. They are usually linked with previous neurons in a fully-connected way and they are the final threshold for predicting.

In general, CNNs have gained a lot of interest in researching the meaning behind the combination of those different layers. The advantages brought by the structure of CNNs include reduced number of parameters and translation invariance.

3. Acceleration Method Taxonomy

Our taxonomy is shown in Figure 3. The philosophy behind the taxonomy is the order from designing, to training a CNN and finally to implementing it on hardware. For the CNN structure, there is redundancy in both weights and the number of bits for representation. For the redundancy in weights, layer decomposition, network pruning, block-circulant projection and knowledge distillation methods can be applied. For the redundancy in representation, using fixed-point representation is the mainstream. Once the structure is decided, CNN adopts training algorithms that are generally used in other neural networks for training process. The most popular training method is gradient decent based back-propagation algorithm. By propagating errors back from output to input and by adjusting weights wired in the network, errors can be reduced to an acceptable degree. The criterion for algorithm optimization is convergence speed with proper stability. Considering that convolutional layers are computationally intensive, we are also interested in the convolution operation complexity. Therefore, we also summarize some efficient convolution methods that are adopted in the CNN. As for the implementation level, the mainstream GPU, FPGA, ASIC are discussed. Recently, people see a promising future for fast implementation of CNN as neuromorphic engineering develops. Some new devices are also presented in this paper. The acceleration approaches of each level is orthogonal and can be combined with those in other levels. By researching such a wide range of methods, we expect to provide a general idea on CNN acceleration, especially for deep CNN, from the perspectives of structure, algorithm, and hardware.

4. Structure Level

Many training and inference process can be accelerated by reducing redundancy in network structures. There is redundancy both in weights and in the way how weights are represented. Two perspectives of acceleration methods will be summarized as follows in terms

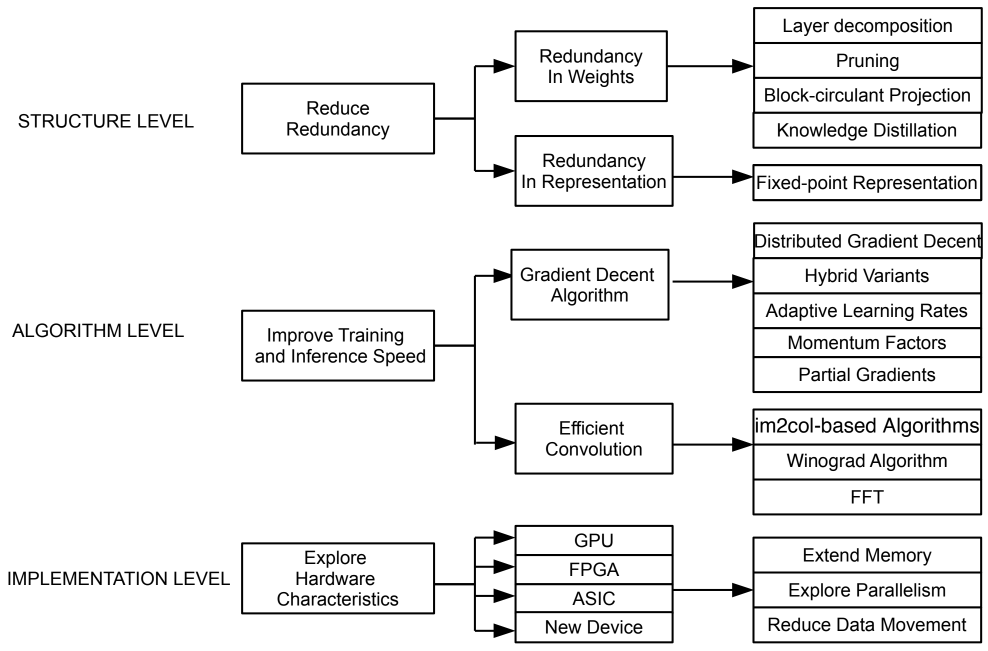


Figure 3: Taxonomy of CNN acceleration methods.

of redundancy in weights and redundancy in representations.

4.1. *Redundancy In Weights*

There is significant redundancy in the parameterization of some neural networks. As Denil *et al.* and Sainath *et al.* observe that some weights learned in networks are correlated with each other, they demonstrate that some of the weights can either be predicted or be unnecessary to learn [45, 46].

4.1.1. *Layer Decomposition*

Low-rank approximation can be adopted to reduce redundancy in weights [47]. For one layer, the input-output relationship can be described by

$$\mathbf{y} = g(\mathbf{x} \cdot \mathbf{W}), \tag{4}$$

where $\mathbf{W}$ is the weight matrix with size $m \times n$. $\mathbf{W}$ can be replaced by the product of two full rank matrices $\mathbf{U} \cdot \mathbf{V}$ with size $m \times r$ and $r \times n$ respectively. The number of parameters in $\mathbf{W}$ can be reduced to $1/d$ if the following inequality holds, $d(mr+rn) < mn$. An efficient low-rank approximation of kernels can be applied in first few convolutional layers of CNN to exploit the linear structure of the over-parameterization within a filter. For example, Denton *et al.* reduce the computation work for redundancy within kernels. It achieves $2 \sim 2.5\times$ speedup with less than 1% drop in classification performance for a single convolutional layer. It uses singular value decomposition method to exploit the approximation of kernels with assumptions that the singular values of the kernels decay rapidly so that the size of the kernels can be reduced significantly [48].

Instead of treating kernel filters as different matrices, kernels in one layer can be treated as a 3D tensor with two spatial dimensions and the third dimension representing channels. Lebedev *et al.* use CP-decomposition for convolutional layers, which achieves $8.5\times$ CPU speedup at the cost of 1% error increase [49]. Tai *et al.* utilize tensor decomposition to remove the redundancy in the convolution kernels, which achieves twice more efficiency of inference for VGG-16 [50]. Wang *et al.* propose to use group sparse tensor decomposition for each convolutional layer, which achieves $6.6\times$ speed-up on PC and $5.91\times$ speed-up on mobile device with less than 1% error rate increase [51]. Tucker decomposition is also used recently to decompose pre-trained weights with fine-tuning afterwards [52, 53].

Weight matrix decomposition method can not only be applied to convolutional layers, but also fully-connected layers. Applying the low-rank approximation to the fully-connected layer weight can achieve a $30 \sim 50\%$ reduction of number of parameters with little loss in accuracy, which is roughly an equivalent reduction in training time [46]. In spite of using two full rank matrices

$$\mathbf{W} = \mathbf{U} \cdot \mathbf{V}, \tag{5}$$

some works have proposed different decomposition forms

$$\mathbf{W} = \mathbf{D}_1 \cdot \mathbf{H} \cdot \mathbf{P} \cdot \mathbf{D}_2 \cdot \mathbf{H} \cdot \mathbf{D}_3, \tag{6}$$

with diagonal matrices $\mathbf{D}_{1,2,3}$ and Hadamard matrix $\mathbf{H}$ [54], and

$$\mathbf{W} = \mathbf{A} \cdot \mathbf{C} \cdot \mathbf{D} \cdot \mathbf{C}^{-1}, \tag{7}$$

with diagonal matrices $\mathbf{A}$, $\mathbf{D}$ and DCT matrix $\mathbf{C}$ [55]. This method can be used during training the CNN, which is very meaningful. The CNN efficiency can be further improved if the training complexity can be reduced as well. Ioannou *et al.* propose to learn some basis small filters that can describe the more complex filters from the scratch. By carefully choosing the initialization status, the new method can be used during training [56]. Wen *et al.* force more weight information into the filters to get more efficient CNNs. With its help, the training process converges faster during the fine-tuning phase. In the experiments, it obtains 2× faster on GPU without accuracy loss [57].

The decomposition technique is layer oriented and can be interleaved with other modules such as ReLU modules in CNN. It can also be applied to the structure of neural networks. Rigamonti *et al.* apply this technique to the general frameworks and reduce the computational complexity by using linear combinations of fewer separable filters [58]. This method can be extended for multiple layers (e.g. > 10) by utilizing low-rank approximation for both weights and input [59]. It can achieve 4× speedup with 0.3% error increase for deep network models VGG-16 by focusing on reducing accumulated error across layers using generalized singular value decomposition.

The methods above can be generalized as layer decomposition for filter weight matrix dimension reduction, while pruning is another method for dimension reduction.

4.1.2. Network Pruning

Network pruning originates as a method to reduce the size and over-fitting of a neural network. As neural network implementation on hardware becomes more popular, it is necessary to reduce the limitation such as its intensive computation and large memory bandwidth requirement. Nowadays, pruning is usually adopted as a method to reduce the network size and to increase the network inference speed so that it can be applied in specific hardware such as embedded systems.

There are many pruning methods in terms of weights, connections, filters, channels, feature maps, and so on. Unlike layer decomposition in which computational complexity is reduced through reducing the total size of layers, selected neurons are removed in pruning. For pruning weights, the unimportant connections of weights with magnitudes smaller than a given threshold are dropped. Experiments are taken on NVIDIA TitanX and GTX980 GPUs, which achieves 9× and 13× parameter reduction for AlexNet and VGG-16 models respectively with no loss of accuracy [60]. Zhou *et al.* incorporate sparse constraints to decimate the number of neurons during training, which reduces the 70% number of neurons without accuracy sacrifice [61]. Besides the method to eliminate least influential neurons, another method is to merge selected and the rest of neurons to maintain diversity in the information. Mariet *et al.* succeed in merging the qualified neurons with unqualified ones and reduce the network complexity [62]. By trading off the training error with the remaining hidden neurons, they achieve 25% reduction of the original number of parameters with 0.04 accuracy reduction in MNIST dataset. Channel pruning method is to eliminate lowly active channels, which means filters are applied in fewer number of channels in each layer. Polyak *et al.* propose a channel-pruning based method Inbound Prune to compress a redundant network. Their experiment is taken on the platform of Samsung Galaxy S6 and it achieves 1.59× speedup [63]. Recently, pruning is combined with other acceleration techniques to achieve speedup. For example, Han *et al.* combine pruning with trained quantization and Huffman coding to deep compress the neural networks in three steps. It achieves 3× layer-wise speedup on fully-connected layer over benchmark on CPU [60].

Some of these pruning methods result in structured sparsity, while others cause unstructured sparsity such as weight-based pruning. Many techniques are proposed to deal with problems of unstructured sparsity being unfriendly to hardware. Wen *et al.* propose a method called Structured Sparsity Learning (SSL) for regularizing compressed structures of deep CNNs and speeding up convolutional computation by group Lasso regularization and locality optimization respectively. It improves convolutional layer computation speed by 5.1× and 3.1× over CPU and GPU [64]. He *el al.* propose a channel pruning method by iteratively reducing redundant channels through solving LASSO and reconstructing the outputs with linear least squares. It achieves 5× speed increase in VGG-16 and 2× speedup in ResNet/Xception [65]. Liu *et al.* also impose channel-based pruning. They use L1 regularization and achieve

20× reduction in model size and 5× reduction in computing operations for VGG model [66]. Li *et al.* prune whole filters as well as their related feature maps and reduce inference cost of VGG-16 by 34% and ResNet-110 by 38% [67]. Their method uses sum of filter's absolute values as a measurement of filter importance, which is filter-based and avoids sparse connectivity. Based on Taylor expansion of cost function between pruning and non-pruning situations, Molchanov *et al.* reduce feature maps from convolutional layers and implement the iterative pruning method in transfer learning setting [68]. ASIC based methods dealing with irregular sparsity are proposed as well and will be discussed in Section 6.3.

4.1.3. Block-circulant Projection

A square matrix could be represented by a one-block-circulant matrix, while a non-squared matrix could be represented by block-circulant matrix. Block-circulant matrix is one of the structured matrices that is usually used in paradigms such as dimension reduction [69], since it can represent an unstructured matrix with a vector. A one-block-circulant matrix is defined as

$$\mathbf{R} = circ(\mathbf{r}) := \begin{bmatrix} r_0 & r_{d-1} & \cdots & r_2 & r_1 \\ r_1 & r_0 & r_{d-1} & \cdots & r_2 \\ \vdots & r_1 & r_0 & \ddots & \vdots \\ r_{d-1} & r_{d-2} & \cdots & r_1 & r_0 \end{bmatrix}, \tag{8}$$

which can be represented by a vector $\mathbf{r} = (r_0, r_1, \ldots, r_{d-1})$. Block-circulant based CNN has been explored nowadays as it has small storage requirements.

Cheng *et al.* apply the circulant matrix in the fully connected layer and achieve significant gain in efficiency with little decrease in accuracy [70]. Yang *et al.* focus on reducing the computational time spent in fully-connected layer by imposing the circulant structure on the weight matrix for dimension reduction with little loss in performance [71]. Ding *et al.* propose to use block-circulant structure in both fully-connected layers and convolutional layers in non-square-matrix situations to further reduce the storage waste. They also mathematically prove that fewer weights in circulant form do not harm the ability of a deep CNN without weight redundancy reduction [72].

4.1.4. Knowledge Distillation

Knowledge distillation is a concept that information obtained from a large complex ensemble neural networks can be utilized to form a compact neural network

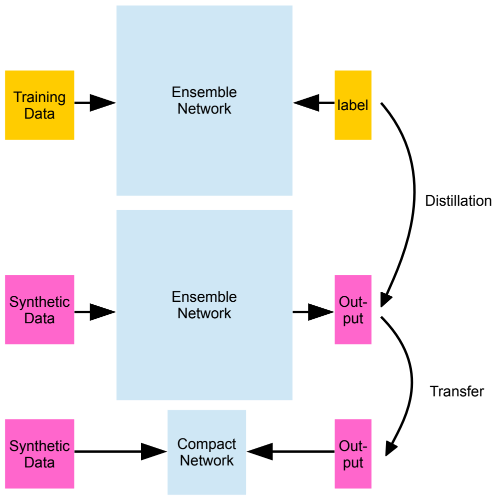


Figure 4: Illustration of knowledge distillation.

[73]. The way that knowledge is transferred can be depicted in the following Figure 4. Information flow from one complex network to a simpler one by training the latter one with data labeled by the former network. By using synthetic data generated from a complex network to train a compact model, it is less likely to cause overfitting and can approximate the functions very well. More importantly, it provides a new perspective for model compression and complicated neural network acceleration.

Synthetic data is very important in succeeding model compression. If it matches well with the true distribution from the functions of a complex model, it usually takes less data for training to mimic it with high-fidelity. Furthermore, the compact model has good generalization characteristics in some missions as it reduces overfitting. Bucilu *et al.* lay a foundation for mimicking a large machine learning model by experimenting three ways to generate pseudo data, which are random, naive bayes estimation, and MUNGE respectively [74]. Some researches propose teacher-student format, which also adopts knowledge distillation concepts with different methods for synthesizing data. For example, Hinton *et al.* compress a deep teacher network into a student network using data combined from teacher network outcome and the true labeled data [75]. The student network can achieve very high accuracy on MNIST dataset with less run time of inference. Romero *et al.* mimic a wider and shallower teacher neural network with a thinner and deeper network called a student network by learning an intermediate representation that is predicted by the teacher network [76]. The depth of the student network ensures its performance, while its thin characteristic reduces the computation complexity.

4.2. Redundancy in Representations

Many weights in neural networks have very small values. For example, the first non-zero digit of many weights occurs in the eighth decimal place, which requires more precise way to record them. Most arithmetic operations in neural networks use 32-floating point representation in order to achieve a good accuracy. As a trade-off, that increases the computation workload and memory size for the neural networks. However, arithmetic operations in fixed-point instead of floating-point can achieve enough good performance for neural networks [77]. A 16-bit fixed-point representation method is proposed by using stochastic rounding for training CIFAR-10 dataset [78]. A further compression of 10-bit dynamic fixed-point is also explored [79]. Han *et al.* quantize pruned CNNs to 8-bit and achieve further storage reduction with no loss of accuracy [60].

For now, representation in one bit is the simplest form. In terms of binarization, there can be three forms, binary input, binary weights of the network, and binary operations. Courbariaux *et al.* propose a BinaryConnect method to use 1-bit fixed-point weights to train a neural network [80]. Rastegari *et al.* come up with a XNOR-Nets with binary weights and binary input fed to convolutional layers [81]. It results in 58× speedup of convolutional operations. Kim *et al.* propose a Bitwise Neural Network, which takes everything as binary such as weights, bias terms, input, output, and basic logic operations instead of floating or fixed-point arithmetic operations [82]. Zhou *et al.* propose to train CNN using binary and stochastically quantized low bit-width gradients and achieve comparable performance as 32-bit counterparts [83]. Hubara *et al.* propose training methods for quantized neural networks that use low precision including 1-bit weights and activations, and replace most arithmetic operations with bit-wise operations [84]. Kim *et al.* compress binary weight CNNs by decomposing kernels into sub-kernels with common parts. They reduce the operation of each image by 47.7% [85]. Ternary CNNs are proposed recently as a more expressive method comparing to binary CNNs, which seeks to achieve a balance between binary networks and full precision networks in terms of compression rate and accuracy [86, 87, 88].

Stochastic computing (SC) is a type of technique that simplifies numerical computations into bit-wise operations by representing continuous values with random bit streams. It provides many benefits for neural networks such as low computation footprint, error tolerance, simple implementation in circuits and better trade-off between time and accuracy [89]. Many works contribute to exploring potential space in optimization and in deep belief networks [90, 91, 92]. Recently it starts to gain attentions in CNN field and regarded as a promising technique for deep CNN implementation on ASIC (Section 6.3) and on embedded portable devices as it can significantly reduce resource consumption with high accuracy.

SC is first adopted in deep CNN by Ren *et al.* with proposed method called SC-DCNN. They design both function blocks and feature extraction blocks that help SC efficiently implemented in deep CNN. It successfully achieves the lowest resource consumption of LeNet5 with optimized configurations among many state-of-the-art software and hardware platforms [93]. Li *et al.* further improve SC based DCNN by introducing normalization in the hardware implementation and dropout in DCNN software training. They design the stochastic normalization circuit by decoupling complex normalization into three units, namely, square and summation, activation and division. Their proposed method improves the SC-based DCNN with 3.26% top-1 accuracy and 3.05% top-5 accuracy [94].

Although errors may accumulate due to representation approximation, its hardware implementation can achieve a much faster speed and lead to less energy consumption.

5. Algorithm Level

In the training process, gradient-based method is widely used in multi-layer feedforward neural networks (FNN), while some other models use analytically determined methods to minimize the cost function [95]. In the forward pass, the output of CNN is calculated, while in the backward pass, weights and bias are adjusted. By reducing the number of iterations to converge, training time can be decreased. Therefore, optimizing gradient decent algorithm is very important for improving the performance in training. For CNN, convolution computation reduces the amount of weights dramatically because it focuses on a local perception field. But repeated mathematic addition and multiplication increase the computation intensity. Therefore, in the forward process, convolution operation workloads are computationally intensive and become constrained for implementation. In the following, we discuss the algorithm optimization of the two directions of data flow, which are gradient-based backward training methods and convolution-based forward inference methods. We summarize distributed gradient descent methods, hybrid variants of the gradient decent and the improvement in terms of self-adaptive learning rates, momentum factors, and partial gradients. We also give an overview on im2col-based algorithms, Winograd based algorithms, and FFT, all of which address the convolution cost problem in CNN.

5.1. *Gradient Decent Optimization*

Gradient decent is one of the most popular algorithms for optimization. It has been largely used in finding global minima for error functions during training neural networks, because it is simple and empirical to implement. The core of mathematical model of gradient decent algorithm is the update rule $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, where the parameters are updated in the opposite direction of the gradient of the error function $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

Distributed gradient decent methods have been proposed to alleviate hardware workload. Take Google training CNN [75] as an example. As illustrated in Figure 5, there are two types of parallelism for the distribution. (a) Replica of CNNs are trained through a server using averaging gradients and different batches of data. Parameters are updated based on all the average gradients, which indicates that new parameters reflect the features from the whole data. (b) For each replica of CNN, it distributes the computation into different cores with different subset of neurons. Its implementation will be introduced in Section 6.1.

Back-propagation algorithm is a form of gradient decent algorithm that is implemented in the neural networks. Some hybrid variants of back-propagation have been proposed in order to take advantage of the benefits from other algorithms. For example, combining it with cuckoo search algorithm can increase the searching speed for optimal solutions [96]. The combination with ant colony algorithm can decrease the computational cost with increased stability of convergence [97]. Pan *et al.* introduce three stages in the back-propagation with genetic algorithms and steepest decent methods combined together to achieve a fast and stable goal [98]. Ding *et al.* use genetic algorithms to optimize the weights of a back-propagation neural network by encoding and thresholding the connection weights [99].

For deep CNN, as errors accumulate layer by layer, the gradient either decays rapidly to zero or increases out of bound. Researchers focus on making changes in error functions, learning rates and incorporating momentum [100] to reduce the derivative vanishing effects and to improve the speed of convergence in the 1990's, while in the recent 7 years, incorporating various factors with momentum factors, introducing self-adaptive learning rates, and using partial gradients are mainstreams to improve the gradient algorithms.

For example, adapting learning rate to parameters with exponentially decaying average of squared gradients leads to a varying learning rate, which depends on each current and past parameter instead of being a constant [101, 102, 103]. Hamid *et al.* incorporate the momentum factor and give control over it, which accelerates the convergence speed especially for oscillating situations in ravine [104]. Nesterov *et al.* have proposed to use partial gradient to update each parameter rather than using the whole gradient [105, 106]. They randomly collect feature dimensions by sampling a block of coordinates and taking partial derivatives over this block, which can dramatically reduce the gradient computation complexity. As a result, it is much faster than the regular stochastic gradient decent method especially for high-dimensional dataset.

For deep neural networks, gradient descent with back-propagation is not guaranteed to find the global minimum of the error function, and is subject to weight vanishing or exploding. The former issue is due to the non-convexity of error functions in neural networks. Some works focus on non-gradient-based methods, such as ant bee colony algorithms and genetic algorithms. They are usually for simple dataset like Boolean dataset and simple neural network structures with one to two hidden layers. In practical, local minimum problem can be leveraged by a deep architecture [20]. The second issue that weight vanishes or explodes when the amount of layers accumulates is still an open problem and has much potential to explore.

5.2. *Feed-forward Efficient Convolution*

Three methods are summarized for the feed-forward efficient convolution including im2col-based algorithm, Winograd based method, and FFT based method, with the most commonly used one being introduced firstly. For the direct convolution in the CNN, convolution kernels slide over the two dimensions of the input and the output is obtained by dot product between the kernels and the input. While for the im2col-based algorithms, the input matrix is linearized into multiple lowered vectors, which can be later efficiently computed [107, 108, 109]. Cho *et al.* further reduce the linearization memory-overhead and improve the computational efficiency by modifying both the lowered vectors and the vectorized kernels [110]. Winograd based methods are to incorporate Winograds minimal filtering algorithms to compute minimal convolution over small filters. Coppersmith Winograd algorithm is known as a fast matrix multiplication algorithm. Winograd based convolution reduces the multiplications by increasing the number of additions and it reduces the memory

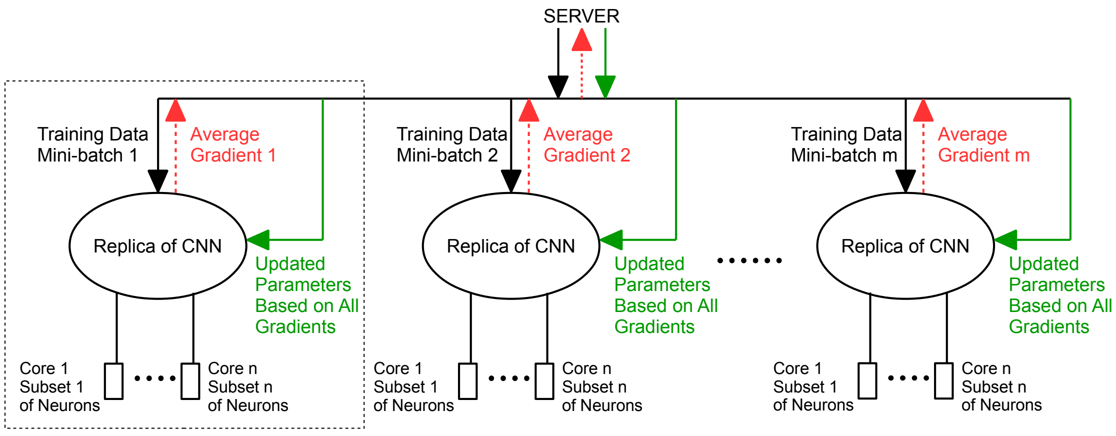

SERVER
Training Data Mini-batch 1
Average Gradient 1
Replica of CNN
Updated Parameters Based on All Gradients
Core 1 Subset 1 of Neurons
Core n Subset n of Neurons
Training Data Mini-batch 2
Average Gradient 2
Training Data Mini-batch m
Average Gradient m

Figure 5: Illustration of CNN distributed system.

consumption on GPU [111, 112]. Winograd's minimal filtering algorithms can help reduce convolution computation at the expense of memory bandwidth. Xiao *et al*. utilize Winograd's minimal filtering theory combined with heterogeneous algorithms for a fusion architecture to mitigate memory bandwidth problem [113].

Based on the experiment that FFT can be applied in MLP to accelerate the first layer inference [114], Mathieu *et al*. first apply FFT on weights of CNN and achieve good performance and speedup for large perceptive areas [115]. For using FFT in CNN, it is necessary to transform back and forth between time domain and frequency domain, which consumes a lot of resources and takes time. Therefore, it needs delicate balance between the benefits of computation in frequency domain and the drawbacks of transforming back and forth. Large perception areas have better performance, which results in limitation in the neural network with small convolution filters. In order to solve this problem, one of the solutions is to train weights directly in frequency domain [116]. Ko *et al*. train the CNNs entirely in the frequency domain with approximate frequency-domain nonlinear operations, sinc interpolation and Hermitian symmetry. By eliminating Fourier transforms at each layer, they achieve significantly training time reduction for CIFAR-10 recognition [117].

6. Implementation Level

Neural networks regain their vigor due to high performance hardware recently. CPU used to be the main stream for implementing machine learning algorithms about twenty years ago, because matrix multiplication and factorization techniques were not popular back then. Nowadays, GPU, FPGA, and ASIC are utilized for accelerating training and predicting process. Besides, much new device technology is proposed to meet requirement for very large models and large training datasets. In the following, hardware based accelerators are summarized in terms of GPU, FPGA, ASIC and frontier new device that is promising for accelerating deep convolutional neural networks.

6.1. GPU

In terms of GPU, clusters of GPUs can accelerate very large neural networks with over one billion parameters in a parallel way. The mainstream of GPU cluster neural networks usually work with distributed SGD algorithms as illustrated in Section 5.1. Many researches further exploit the parallelism and make efforts on communication among different clusters. For example, Baidu Heterogeneous Computing Group uses two types of parallelism called model-data parallelism and data parallelism to extend CNN architectures to 36 servers, each with 4 NVIDIA Tesla K40m GPUs and 12GB memory. The strategies include butterfly synchronization and lazy update, which makes good use of overlapping in computation and communication [118]. Coates *et al*. propose a clustering of GPU servers using Commodity Off-The-Shelf High Performance Computing (COTS HPC) technology and high-speed communication infrastructure for parallelism in distributed gradient decent algorithm, which reduces 98% number of machines used for training [119]. In terms of non-distributed SGD algorithms, Imani *et al*. propose a nearest content addressable memory block called NNCAM, which stores highly frequent patterns for reusing. It accelerates CNNs over general purpose GPU with 40% speedup [120].

6.2. FPGA

There are many parallelism levels in hardware acceleration, such as coarse-grain, medium-grain, fine-grain, and massive [121]. FPGA outperforms in terms of its fine grain and coarse grain reconfiguration ability and its hierarchical storage structure and scheduling mechanism can be optimized flexibly. Flexible hierarchical memory systems can support complex data access mode of CNN. It is often used to improve the efficiency of on-chip memory and to reduce the energy consumption.

Peemen *et al.* experiment on Virtex 6 FPGA board and show that the accelerator design can achieve 11× speedup with very complicated address mapping of data access [122]. Zhang *et al.* take data reuse, parallel processing, and off-chip memory bandwidth into consideration in FPGA accelerator. The accelerator achieves 17.42× faster speed than CPU in AlexNet CNN architecture [123]. Martnez *et al.* take advantage of the FPGA reconfiguration characteristics by unfolding the loop execution on different cascading stages. As the number of multipliers for convolution increases, the proposed method can achieve 12 GOPS at most [124]. A hardware acceleration method for CNN is proposed by combining fine grain in operator level parallelism and coarse grain parallelism. Compared with 4xIntel Xeon 2.3 GHz, 1.35 GHz C870, and a 200 MHz FPGA, the proposed design achieves a 4× to 8× speed boost [125]. Wang *et al.* propose an on-chip memory design called Memsqueezer that can be implemented on FPGA. They shrink the memory size by compressing data, weights, and intermediate data from the perspectives of hardware, which achieves 80% energy reduction compared with conventional buffer designs [126]. Zhang *et al.* design an FPGA accelerator engine called Caffeine that decreases underutilized memory bandwidth. It reorganizes the memory access according to their proposed matrix-multiplication representation applied to both convolutional layers and fully-connected layers. Caffeines implementation on Xilinx KU060 and Virtex 7690t FPGA achieves very high peak performance of 365 GOPS and 636 GOPS respectively [127]. Rahman *et al.* present a 3D array architecture, which can benefit all layers in CNNs. With optimization of on-chip buffer sizes for FPGAs, it can outperform the state-of-the-art solutions by 22% in terms of MAC [128]. Alwani *et al.* explore the design space of dataflow across multiple convolutional layers, where a fused layer accelerator is designed that reduces feature map data transfer from and to off-chip memory [129].

6.3. ASIC

For ASIC design, despite of using methods in structure level such as block-circulant projection in Section 4.1.3 and SC in Section 4.2, to improve the design in implementation level, memory can be expanded and locality can be increased to reduce data transporting within systems for deep neural network accelerating. Tensor Processing Unit (TPU) is designed for low precision computation with high efficiency. It uses a large on-chip memory of 28MiB to execute the neural network applications, which can achieve at most 30× faster speed than an Nvidia K80 GPU [10]. TETRIS is an architecture using 3D memory proposed by Gao *et al.* It saves more area for processing elements and leaves more space for accelerator design [130].

Luo *et al.* create an architecture of 64-chip system that minimizes data moving between synapses and neurons by storing them closely. It reduces the burden on external memory bandwidth and achieves a speedup of 450× over a GPU with 150× energy reduction [131]. Wang *et al.* propose to group adjacent process engines (PEs) into dual-channel PEs called Chain-NN to mitigate huge amount of data movements. They simulate it under TSMC 28nm process and achieve a peak throughput of 806.4 GOPS in AlexNet [132]. Single instruction multiple data (SIMD) processors are used on a 32-bit CPU to design a system targeted for ASIC synthesis to perform real-time detection, recognition and segmentation of mega-pixel images. They optimize the operation in CNN with available parallelism in hardware. The ASIC implementations outperform the CPU conventional methods in terms of frames/s [133].

Recently, some ASIC designs target at sparse networks with irregularity. For example, Zhang *et al.* propose an accelerator called Cambricon-X that can reach 544 GOP/s in $6.38mm^2$ [134]. It consists an Indexing Module, which can efficiently schedule processing elements that store irregular and compressed synapses. Kwon *et al.* design a reconfigurable accelerator called MAERI to adapt various layer dataflow patterns. They can efficiently utilize compute resources and provides 6.9× speedup at 50% sparsity [135]. Network pruning could induce sparsity and irregularity as discussed in Section 4.1.2. With such designs, better performance is expected to achieve when combined.

6.4. New Devices

As new device technology and circuits arise, deep convolutional neural networks can be potentially accelerated by orders of magnitude. In terms of new device, very large scale integration systems are explored to mimic complex biological neuron architectures.

Some of them are in their theoretical demonstration state for training deep neural networks. For example, Gokmen and Vlasov from IBM research center propose a resistive processing unit (RPU) device, which can both store and compute parameters in this unit. It has extremely high processing speed with 30000× higher than state-of-the-art microprocessors (84000 GigaOps/s/W) [136]. As neuromorphic engineering develops, more new devices emerge to handle high frequency and high volume information transformation through synapses. Some are in theoretical state that have not been implemented on neural networks for classification and recognition, such as nano-scale phase change device [137] and ferroelectric memristors [138].

Resistive memories are treated as one of the promising solutions for deep neural network accelerations due to its nonvolatility, high storage density, and low power consumption [139]. Its architecture mimics neural networks, where weight storage and computation can be done simultaneously [140, 141]. As CMOS memories become larger, its scale becomes limited. Therefore, besides the main stream CMOS based memory, nonvolatile memory becomes more popular in storing weights, such as resistive random access memory (RRAM) [142, 143, 144, 145] and spin-transfer torque random access memory (STT-RAM) [146].

Memristor crossbar array structures can deal with computational expensive matrix multiplication and have been explored in CNN hardware implementations. For example, Hu *et al.* develop a Dot-Product Engine (DPE) utilizing memristor crossbar, which achieves 1000× to $10,000\times$ speed-efficiency product compared with a digital ASIC [147]. Xia *et al.* address energy consumption problem between crossbars and ADC/DAC and can save more than 95% energy with similar accuracy of CNN [148]. Ankit *et al.* propose a hierarchical reconfigurable architecture with memristive crossbar arrays called RESPARC, which is 15× more energy efficient and has 60× more throughput for deep CNNs [149].

In general, for any CNN hardware implementation, there are a lot of potential solutions to be explored in design space. It is not trivial to design a general hardware architecture that can be applied to every CNN, especially when limitations on computation resource and memory bandwidth are considered.

7. Discussion

Researches have different flavors over dataset, model, and implementation platforms. Many datasets and models are treated as benchmarks based on previous researches. But different benchmarks and their combinations make it difficult to compare the method in one level with one in other levels. In the following discussion, we constrain our comparison and analysis within each level.

7.1. Structure Level

We have summarized methods of layer decomposition and pruning in Table 3. Some of the layer decomposition and pruning methods focus on inference, because pre-trained CNNs are required before applying the corresponding methods. It is a limitation comparing to other acceleration methods. For example, some large scale networks still need training for weeks or months before layer decomposition method implementation [47, 48]. Pruning by sparsified weights and their connections require pre-training on the original full model and fine-tuning [150, 151].

Many layer decomposition and pruning methods are layer-wised and optimized for specific layer when they are first time proposed. For example, Sainath *et al.* demonstrate significant reduction in parameters in the output softmax layer [46]. Mariet *et al.* successfully prune 25% of the parameters with good performance in the fully connected layer [62]. Denton *et al.* successfully reduce a large magnitude number of parameters in the convolutional layer [48]. As these methods focus on different types of layers, there is exploration space about how to combine them for further acceleration.

After reducing redundancy in representation, the size of neural networks can reduce dramatically. However, specific hardware is required to achieve a speedup in training and testing, since currently most GPUs are improved to suit for floating-point performance. For example, using BinaryConnect method [152] to train a Torch7 frame ConvNet on GPU takes more time. The time complexity can be reduced theoretically by 60% if using dedicated hardware.

We have summarized methods of reducing redundancy in representation in Table 4 and Table 5. The Note column in these two tables provide important information that needs to be distinguished among different methods. Low-bit representation methods are targeted for both large and small models. Various bit-width representation results in different performance dependent on different models and datasets. Many experiments are conducted based on small datasets with low resolution (e.g. 32×32) such as CIFAR10, and usually achieve less than 5% error rate increase. For image classification at large scale (e.g. ImageNet), low-bit representation method is difficult to achieve the same performance as that for small dataset.

	Method	Target Layer	Pre-training	Performance
Layer decomposition	[47]	Convolutional layers	required	2.5× speedup with no loss in accuracy
	[48]	Convolutional layers	required	2× speedup with < 1% accuracy drop
	[52]	Whole network	required	1.09× reduction in weights & 4.93× speedup in VGG-16
	[56]	Convolutional layers	not required	76% reduction in weights in VGG-11
Pruning	[150]	Whole network	required	prune 90% parameters of the convolutional kernels
	[151]	Whole network	required	prune 13× parameters in VGG-16
	[64]	Whole network	not required	5.1× (CPU) & 3.1× (GPU) speedup in convolutional layers
	[67]	Whole network	required	34% inference FLOP reduction in VGG-16

Table 3: Layer decomposition and pruning methods analysis

Bit-width	Method	Model	Error rate increase	Note
Binary	BinaryConnect [80]	Self-designed (e.g. 6C-3S-2F-L2SVM)	∼ 2% error rate drop	Binary weights during training and testing
	Binarynet [152]	Self-designed	10.15% absolute error rate	Binary weights & activations during forward pass
Ternary	TWN [86]	VGG-7	< 1%	Ternary weights in forward & backward pass
	Ternary Connect [87]	6C-1F-1softmax	∼ 3% error rate drop	Ternary weights during training
	TNN [88]	VGG-variant	12.11% absolute error rate	Teacher-student approach based ternary input & activations during training
Others	12/14/16-bit [78]	3C-3S-1softmax	< 5%	Fixed-point number representation with stochastic rounding
	10-bit [79]	Maxout networks	< 5%	Dynamic 10-bit fixed point precision during training

Table 4: Representation Reduction Methods (CIFAR10)
C: convolutional layer, S: subsampling layer, F: fully-connected layer

7.2. Algorithm Level

7.2.1. Information for Updating

According to the cost function of gradient descent algorithm, there are two categories, which are first-order derivatives gradient descent and second-order derivatives gradient decent. Compared with first-order derivatives based gradient decent algorithm, the second-order one is faster to converge. But it is more complex to

Bit-width	Method	Model	Error rate increase	Note
Binary	QNN [84]	GoogLeNet, AlexNet	$> 10\%$	Binary weights & activations during training and testing
	XNOR-net [81]	AlexNet, ResNet, GoogLeNet-variation	$> 10\%$	Binary weights & input to convolutional layers
	BWN [81]	AlexNet, ResNet, GoogLeNet-variation	$< 10\%$	Binary weights in forward & backward pass
	DoReFa-Net [83]	AlexNet	around 10%	Binary weights & 2-bit activations & 6-bit gradients
Ternary	TWN [86]	ResNet	$< 5\%$	Ternary weights in forward & backward pass

Table 5: Representation Reduction Methods (ImageNet).

utilize the second order information, which makes it prohibitive in practice for deep large neural networks. Therefore, more emphasis has been put on how to approximate the Hessian matrices, which consists of the second-order derivatives for simplicity [153].

7.2.2. *Data for Training*

According to the update amount of data, there are three variants of the algorithms, which are batch, mini-batch, and online. The batch method uses whole dataset to update the gradient in one iteration. The mini-batch uses randomly picked small amount of data to update the gradient while the online method uses new incoming subset of data once to update the gradient and stops at any time.

For batch gradient decent, it is guaranteed to converge to the global minimum for convex surfaces. But it can be very slow and requires very large memory storage. The mini-batch can avoid redundant gradient computation using shuffled examples. As a result, it usually shoots the minimum faster than the batch gradient. With delicate picked learning rate, its fluctuation performance decreases. The online method can be used for designing a light-weight algorithm in terms of memory and speed for handling a stream of data. Since the data is updated frequently, it can be used to predict the most recent state of the trend. But as data is discarded after gradient update, online method is considered to be more difficult and unreliable [154].

7.2.3. *Asynchronous Updating*

Asynchronous training algorithms help improve the efficiency on large-scale clusters of machines for distributed training and inference. Proposed asynchronous stochastic gradient decent algorithms such as Downpour SGD and AASGD help improve neural network performance in classification problems with a large amount of high dimensional data. But more attention is needed on communication among different workers in the clusters, since suboptimal communication can result in parameters diverging [155, 156].

7.2.4. *From Frequency Perspective*

Table 6 summarizes methods of FFT based CNNs. The concept of implementing CNN in frequency is to replace convolution operation in time domain with multiplication in frequency domain. It takes time to transform back and forth. As a result, it performs well on large feature maps. Development is made to suit for small feature maps such as training network directly in frequency domain. Compared with other algorithms, FFT method requires additional memory for padding the filters to the same size of the input and storing frequency domain intermediate results. This leads to a trade-off for hardware implementation. On one hand, it can take use of power in GPU parallelism to speedup convolution computation dramatically. On the other hand, more delicate GPU memory allocation is required due to limit memory.

Method	Platform	Feature	Additional Memory	Complexity of Fourier Transform & Inverse	Complexity of Add & Mul in Frequency Domain & Extra Complexity
Mathieu [115]	GeForce GTX Titan GPU	Perform convolutions as products in frequency domain	Yes	$(2C \cdot n^2 \log n)(S \cdot f + f' \cdot f + S \cdot f')$	$4S \cdot f' \cdot f \cdot n^2$
Rippel [157]	Xeon Phi coprocessor	Pooling in frequency domain	Yes		
Ko [117]	ASIC	Train the network entirely in frequency domain	No	$2Cn^2 \log n(S \cdot f)$	$(3/2 \cdot (n^2 - \alpha) + \alpha) \cdot S \cdot f' \cdot f + 3/2 \cdot n^2 \cdot k^2 \cdot f' \cdot f$
S: mini-batch size, f: input feature map depth, f': output feature map depth, n: feature map dimension k: kernel dimension, C: hidden constant in the O notation, α: 1 for odd and 4 for even n					

Table 6: FFT based method analysis.

Method	Platform	Memory	Frequency	Performance
Minwa [118]	GPUs	6.9 TB host memory & 1.7 TB device memory	-	0.6 PFlops single precision at peak
Roofline-model-based accelerator [123]	VC707 FPGA	-	100MHz	61.62 GFLOPS
Caffeine [127]	Xilinx KU060 FPGA	-	200MHz	365 GOPS
ICAN accelerator [128]	Virtex-7 FPGA	Memory bandwidth 6.2 GB/s	160MHz	147.82 GOPS
Dadiannao [131]	ASIC	36MB node eDRAM	606 MHz	2.09 TeraOps/s of a node at peak
Chain-NN [132]	ASIC	352KB on-chip memory	700MHz	806.4 GOPS at peak
Cambricon-X [134]	ASIC	56KB on-chip SRAM	1 GHz	544 GOPS

Table 7: Performance comparison among GPU, FPGA, and ASIC.

7.3. *Implementation Level*

As CNN becomes more and more complex, general purpose processors cannot exploit the inherent parallelism for matrix or tensor operations and therefore becomes bottleneck when performing large deep convolutional neural networks. Various designs for accelerating network training and inference have been proposed based on GPU, ASIC, and FPGA. Table 7 gives a performance comparison of different methods implementing on GPU, FPGA, and ASIC. Figure 6 shows the comparison among CPU, GPU, FPGA, and ASIC in terms of power and throughput [158, 159, 160, 161, 162].

GPU supports several teraFLOPS throughput and large memory access, but consumes a lot of energy. In terms of economy, GPU costs to set up for large deep convolutional neural networks.

Comparing to GPU, ASIC is specialized hardware and can be delicately designed to maximize its benefits such as power-efficiency and large throughput in CNN implementation. However, once CNN algorithms are

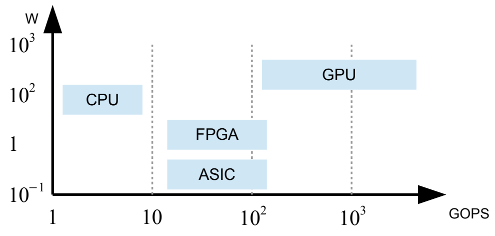


Figure 6: Power and throughput among CPU, GPU, FPGA, and ASIC.

implemented on ASIC, it is difficult to change the hardware design. On the other hand FPGA is easy to be programmed and reconfigured. It is more convenient for prototyping.

Compared with GPU, FPGA throughput is tens of gigaFLOPS and it has limited memory access. In addition, it does not support floating-point natively. But it is more power-efficient. Due to its limited memory access, many proposed methods are focused on accelerating inference time of neural network since inference process requires less memory access comparing to training process. Others are emphasized on external memory optimization for large neural network acceleration. Different models need different hardware optimization and even for the same model, different designs result in quite various acceleration performance [60]. In terms of economy, FPGA is reconfigurable and is easier to evolve hardware, frameworks and software. Especially for various models of neural networks, its flexibility shortens design cycle and costs less.

8. Conclusion

In this paper, we have summarized recent advances in CNN acceleration methods from all structure level, algorithm level, and implementation level. In structure level, CNN is compressed without losing significant accuracy since there is redundancy in most of the CNN architectures. For training algorithms, besides convergence speed, convolution calculation is also an important factor for CNN. FFT method introduces a frequency perspective for training neural networks. In implementation level, characteristics for different hardware such as FPGA and GPU are explored combined with CNN features. CNN performs better in computer vision field as its structure goes deeper and the amount of data becomes larger, which makes it time consuming and computationally expensive. It is imperative and necessary to accelerate CNN for its further implementation in life. For now, there is no generalized evaluation system to test the acceleration performance for comparison among different methods in different levels. Researches use case by case dataset benchmark and different criterion in each level. Therefore, it is challenging in acceleration performance evaluation as well.

Acknowledgement

This research work was partly supported by the Natural Science Foundation of China and Jiangsu (Project No. 6171101145, 61671148, BK20161147), the Research and Innovation Program for Graduate Students in Universities of Jiangsu Province (Grant No. SJLX16_0081, SJCX17_0048), and the Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

References

[1] S. Yu, S. Jia, C. Xu, Convolutional neural networks for hyperspectral image classification, Neurocomputing 219 (2017) 88–98.
[2] C. Farabet, C. Couprie, L. Najman, Y. LeCun, Learning hierarchical features for scene labeling, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) (2013) 1915–1929.
[3] K. Wang, C. Gou, N. Zheng, J. M. Rehg, F. Y. Wang, Parallel vision for perception and understanding of complex scenes: methods, framework, and perspectives, Artificial Intelligence Review (2016) 1–31.
[4] P. Qin, W. Xu, J. Guo, An empirical convolutional neural network approach for semantic relation classification, Neurocomputing 190 (2016) 1–9.
[5] B. Yu, D. Z. Pan, T. Matsunawa, X. Zeng, Machine learning and pattern matching in physical design, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2015, pp. 286–293.
[6] L. Theis, W. Shi, A. Cunningham, F. Huszár, Lossy image compression with compressive autoencoders, in: International Conference on Learning Representations (ICLR), 2017.
[7] M. Zhang, T. Chen, X. Shi, P. Cao, Image arbitrary-ratio down- and up-sampling scheme exploiting dct low frequency components and sparsity in high frequency components, IEICE Transactions on Information and Systems 99 (2) (2016) 475–487.
[8] T. Chen, M. Zhang, J. Wu, C. Yuen, Y. Tong, Image encryption and compression based on kronecker compressed sensing and elementary cellular automata scrambling, Optics & Laser Technology 84 (2016) 118–133.
[9] J. A. A. Jothi, V. M. A. Rajam, A survey on automated cancer diagnosis from histopathology images, Artificial Intelligence Review 48 (1) (2017) 31–81.
[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., In-datacenter performance analysis of a tensor processing unit, in: IEEE/ACM International Symposium on Computer Architecture (ISCA), 2017, pp. 1–12.
[11] NVDLA, http://nvdla.org.
[12] https://www.intelnervana.com.

[13] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, M. S. Lew, Deep learning for visual understanding: A review, Neurocomputing 187 (2016) 27–48.

[14] M. Ghayoumi, A quick review of deep learning in facial expression, Journal of Communication and Computer 14 (2017) 34–38.

[15] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, P. Campoy, A review of deep learning methods and applications for unmanned aerial vehicles, Journal of Sensors 2017.

[16] J. Zhang, C. Zong, Deep neural networks in machine translation: An overview, IEEE Intelligent Systems 30 (5) (2015) 16–25.

[17] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, S. Jain, Machine translation using deep learning: An overview, in: International Conference on Computer, Communications and Electronics (Comptelix), 2017, pp. 162–167.

[18] Z. H. Ling, S. Y. Kang, H. Zen, A. Senior, M. Schuster, X.-J. Qian, H. M. Meng, L. Deng, Deep learning for acoustic modeling in parametric speech generation: A systematic review of existing techniques and future trends, IEEE Signal Processing Magazine 32 (3) (2015) 35–52.

[19] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[20] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[21] X. Du, Y. Cai, S. Wang, L. Zhang, Overview of deep learning, in: Youth Academic Annual Conference of Chinese Association of Automation (YAC), IEEE, 2016, pp. 159–164.

[22] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[23] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Conference on Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105.

[24] M. Lin, Q. Chen, S. Yan, Network in network, arXiv preprint arXiv:1312.4400.

[25] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations (ICLR), 2015.

[26] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[28] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[29] N. Aloysius, M. Geetha, A review on deep convolutional neural networks, in: Communication and Signal Processing (ICCSP), 2017 International Conference on, IEEE, 2017, pp. 0588–0592.

[30] A. A. M. Al-Saffar, H. Tao, M. A. Talab, Review of deep convolution neural network in image classification, in: Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), 2017 International Conference on, IEEE, 2017, pp. 26–31.

[31] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural computation 29 (9) (2017) 2352–2449.

[32] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, New York, 2006.

[33] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: International Conference on Machine Learning (ICML), 2010, pp. 807–814.

[34] A. L. Maas, A. Y. Hannun, A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: International Conference on Machine Learning (ICML), 2013.

[35] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.

[36] B. Xu, N. Wang, T. Chen, M. Li, Empirical evaluation of rectified activations in convolutional network, in: International Conference on Machine Learning Workshop, 2015.

[37] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), in: International Conference on Learning Representations (ICLR), 2016.

[38] P. Sermanet, S. Chintala, Y. LeCun, Convolutional neural networks applied to house numbers digit classification, in: IEEE International Conference on Pattern Recognition (ICPR), 2012, pp. 3288–3291.

[39] M. D. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, in: International Conference on Learning Representations (ICLR), 2013.

[40] D. Yu, H. Wang, P. Chen, Z. Wei, Mixed pooling for convolutional neural networks, in: International Conference on Rough Sets and Knowledge Technology, Springer, 2014, pp. 364–375.

[41] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (9) (2015) 1904–1916.

[42] O. Rippel, J. Snoek, R. P. Adams, Spectral representations for convolutional neural networks, in: Conference on Neural Information Processing Systems (NIPS), 2015, pp. 2449–2457.

[43] Y. Gong, L. Wang, R. Guo, S. Lazebnik, Multi-scale orderless pooling of deep convolutional activation features, in: European Conference on Computer Vision (ECCV), Springer, 2014, pp. 392–407.

[44] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.

[45] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al., Predicting parameters in deep learning, in: Conference on Neural Information Processing Systems (NIPS), 2013, pp. 2148–2156.

[46] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, B. Ramabhadran, Low-rank matrix factorization for deep neural network training with high-dimensional output targets, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 6655–6659.

[47] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, in: British Machine Vision Conference (BMVC), 2014.

[48] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Conference on Neural Information Processing Systems (NIPS), 2014, pp. 1269–1277.

[49] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up convolutional neural networks using fine-tuned cp-decomposition, in: International Conference on Learning Representations (ICLR), 2014.

[50] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al., Convolutional neural networks with low-rank regularization, arXiv preprint arXiv:1511.06067.

[51] P. Wang, J. Cheng, Accelerating convolutional neural networks for mobile applications, in: ACM International Multimedia Conference (MM), 2016, pp. 541–545.

[52] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, in: International Conference on Learning Representations, 2016.

[53] H. Ding, K. Chen, Y. Yuan, M. Cai, L. Sun, S. Liang, Q. Huo, A compact CNN-DBLSTM based character model for offline handwriting recognition with tucker decomposition, in: IAPR International Conference on Document Analysis and Recognition (ICDAR), Vol. 1, IEEE, 2017, pp. 507–512.

[54] Q. Le, T. Sarlós, A. Smola, Fastfood-approximating kernel expansions in loglinear time, in: International Conference on Machine Learning (ICML), Vol. 85, 2013.

[55] M. Moczulski, M. Denil, J. Appleyard, N. de Freitas, ACDC: A structured efficient linear layer, in: International Conference on Learning Representations (ICLR), 2016.

[56] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, A. Criminisi, Training CNNs with low-rank filters for efficient image classification, in: International Conference on Learning Representations (ICLR), 2016.

[57] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, H. Li, Coordinating filters for faster deep neural networks, in: IEEE International Conference on Computer Vision (ICCV), 2017.

[58] R. Rigamonti, A. Sironi, V. Lepetit, P. Fua, Learning separable filters, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[59] X. Zhang, J. Zou, K. He, J. Sun, Accelerating very deep convolutional networks for classification and detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 38 (10) (2016) 1943–1955.

[60] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, in: International Conference on Learning Representations (ICLR), 2016.

[61] H. Zhou, J. M. Alvarez, F. Porikli, Less is more: Towards compact CNNs, in: European Conference on Computer Vision (ECCV), Springer, 2016, pp. 662–677.

[62] Z. Mariet, S. Sra, Diversity networks, in: International Conference on Learning Representations (ICLR), 2016.

[63] A. Polyak, L. Wolf, Channel-level acceleration of deep face representations, IEEE Access 3 (2015) 2163–2175.

[64] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: Conference on Neural Information Processing Systems (NIPS), 2016, pp. 2074–2082.

[65] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: IEEE International Conference on Computer Vision (ICCV), Vol. 2, 2017, p. 6.

[66] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2755–2763.

[67] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, in: International Conference on Learning Representations (ICLR), 2017.

[68] A. Vedaldi, K. Lenc, Matconvnet: Convolutional neural networks for MATLAB, in: ACM International Multimedia Conference (MM), 2015, pp. 689–692.

[69] S. Oymak, C. Thrampoulidis, B. Hassibi, Near-optimal sample complexity bounds for circulant binary embedding, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 6359–6363.

[70] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, S. F. Chang, An exploration of parameter redundancy in deep networks with circulant projections, in: IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2857–2865.

[71] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, Z. Wang, Deep fried ConvNets, in: IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1476–1483.

[72] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, et al., CirCNN: accelerating and compressing deep neural networks using block-circulant weight matrices, in: IEEE/ACM International Symposium on Microarchitecture (MICRO), 2017, pp. 395–408.

[73] R. Caruana, A. Niculescu Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: International Conference on Machine Learning (ICML), 2004, p. 18.

[74] C. Bucilu, R. Caruana, A. Niculescu Mizil, Model compression, in: ACM International Conference on Knowledge Discovery and Data Mining (KDD), 2006, pp. 535–541.

[75] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531.

[76] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, in: International Conference on Learning Representations (ICLR), 2015.

[77] D. Hammerstrom, A VLSI architecture for high-performance, low-cost, on-chip learning, in: International Joint Conference on Neural Networks (IJCNN), IEEE, 1990, pp. 537–544.

[78] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision., in: International Conference on Machine Learning (ICML), 2015, pp. 1737–1746.

[79] M. Courbariaux, Y. Bengio, J. P. David, Training deep neural networks with low precision multiplications, arXiv preprint arXiv:1412.7024.

[80] M. Courbariaux, Y. Bengio, J. P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Conference on Neural Information Processing Systems (NIPS), 2015, pp. 3123–3131.

[81] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-Net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision (ECCV), Springer, 2016, pp. 525–542.

[82] M. Kim, P. Smaragdis, Bitwise neural networks, in: International Conference on Machine Learning (ICML), 2016.

[83] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients, arXiv preprint arXiv:1606.06160.

[84] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, arXiv preprint arXiv:1609.07061.

[85] H. Kim, J. Sim, Y. Choi, L.-S. Kim, A kernel decomposition architecture for binary-weight convolutional neural networks, in: ACM/IEEE Design Automation Conference (DAC), 2017, p. 60.

[86] F. Li, B. Zhang, B. Liu, Ternary weight networks, arXiv preprint arXiv:1605.04711.

[87] Z. Lin, M. Courbariaux, R. Memisevic, Y. Bengio, Neural networks with few multiplications, in: International Conference on Learning Representations (ICLR), 2016.

[88] H. Alemdar, V. Leroy, A. Prost-Boucle, F. Pétrot, Ternary neural networks for resource-efficient AI applications, in: International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2547–2554.

[89] B. D. Brown, H. C. Card, Stochastic neural computation. i. computational elements, IEEE Transactions on Computers 50 (9) (2001) 891–905.

[90] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, Y. Wang, Structural design optimization for deep convolutional neural networks using stochastic computing, in: IEEE/ACM Proceed-

ings Design, Automation and Test in Eurpoe (DATE), 2017, pp. 250–253.

[91] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, K. Choi, Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks, in: ACM/IEEE Design Automation Conference (DAC), 2016, pp. 124:1–124:6.

[92] Y. Ji, F. Ran, C. Ma, D. J. Lilja, A hardware implementation of a radial basis function neural network using stochastic logic, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2015, pp. 880–883.

[93] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, B. Yuan, SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing, in: ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 405–418.

[94] J. Li, Z. Yuan, Z. Li, A. Ren, C. Ding, J. Draper, S. Nazarian, Q. Qiu, B. Yuan, Y. Wang, Normalization and dropout for stochastic computing-based deep convolutional neural networks, Integration, the VLSI Journal.

[95] G. Li, P. Niu, X. Duan, X. Zhang, Fast learning network: a novel artificial neural network with a fast learning speed, Neural Computing and Applications 24 (7-8) (2014) 1683–1695.

[96] N. M. Nawi, A. Khan, M. Z. Rehman, A new back-propagation neural network optimized with cuckoo search algorithm, in: International Conference on Computational Science and Its Applications, Springer, 2013, pp. 413–426.

[97] Y. P. Liu, M. G. Wu, J. X. Qian, Evolving neural networks using the hybrid of ant colony optimization and bp algorithms, in: International Symposium on Neural Networks, Springer, 2006, pp. 714–722.

[98] S. T. Pan, M. L. Lan, An efficient hybrid learning algorithm for neural network–based speech recognition systems on FPGA chip, Neural Computing and Applications 24 (7-8) (2014) 1879–1885.

[99] S. Ding, C. Su, J. Yu, An optimizing bp neural network algorithm based on genetic algorithm, Artificial Intelligence Review 36 (2) (2011) 153–162.

[100] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985).

[101] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (Jul) (2011) 2121–2159.

[102] M. D. Zeiler, ADADELTA: an adaptive learning rate method, arXiv preprint arXiv:1212.5701.

[103] D. Kingma, J. Ba, Adam: A method for stochastic optimization, Journal of Machine Learning Research (2015) 1–13.

[104] N. A. Hamid, N. M. Nawi, R. Ghazali, M. N. M. Salleh, Accelerating learning performance of back propagation algorithm by using adaptive gain together with adaptive momentum and adaptive learning rate on classification problems, in: International Conference on Ubiquitous Computing and Multimedia Applications, Springer, 2011, pp. 559–570.

[105] Y. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems, SIAM Journal on Optimization 22 (2) (2012) 341–362.

[106] P. Richtárik, M. Takáč, Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function, Mathematical Programming 144 (1-2) (2014) 1–38.

[107] cuBLAS, http://docs.nvidia.com/cuda/cublas.

[108] MLK, https://software.intel.com/en-us/intel-mkl.

[109] OpenBLAS, http://www.openblas.net.

[110] M. Cho, D. Brand, Mec: Memory-efficient convolution for deep neural network, in: International Conference on Machine Learning (ICML), 2017, pp. 815–824.

[111] A. Lavin, S. Gray, Fast algorithms for convolutional neural networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4013–4021.

[112] H. Park, D. Kim, J. Ahn, S. Yoo, Zero and data reuse-aware fast convolution for deep neural networks on GPU, in: International Conference on Hardware/Software Codesign and System Synthesis, 2016, pp. 1–10.

[113] Q. Xiao, Y. Liang, L. Lu, S. Yan, Y.-W. Tai, Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas, in: ACM/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

[114] S. Ben Yacoub, B. Fasel, J. Luettin, Fast face detection using MLP and FFT, in: International Conference on Audio and Video-based Biometric Person Authentication, 1999, pp. 31–36.

[115] M. Mathieu, M. Henaff, Y. LeCun, Fast training of convolutional networks through ffts, arXiv preprint arXiv:1312.5851.

[116] T. Brosch, R. Tam, Efficient training of convolutional deep belief networks in the frequency domain for application to high-resolution 2D and 3D images, Neurocomputing.

[117] J. H. Ko, B. Mudassar, T. Na, S. Mukhopadhyay, Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation, in: ACM/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

[118] R. Wu, S. Yan, Y. Shan, Q. Dang, G. Sun, Deep image: Scaling up image recognition, arXiv preprint arXiv:1501.02876 7 (8).

[119] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, N. Andrew, Deep learning with cots hpc systems, in: International Conference on Machine Learning (ICML), 2013, pp. 1337–1345.

[120] M. Imani, D. Peroni, Y. Kim, A. Rahimi, T. Rosing, Efficient neural network acceleration on GPGPU using content addressable memory, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2017, pp. 1026–1031.

[121] N. Izeboudjen, C. Larbes, A. Farah, A new classification approach for neural networks hardware: from standards chips to embedded systems on chip, Artificial Intelligence Review 41 (4) (2014) 491–534.

[122] M. Peemen, A. A. Setio, B. Mesman, H. Corporaal, Memory-centric accelerator design for convolutional neural networks, in: IEEE International Conference on Computer Design (ICCD), 2013, pp. 13–19.

[123] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, in: ACM Symposium on FPGAs, 2015, pp. 161–170.

[124] J. J. Martínez, J. Garrigós, J. Toledo, J. M. Ferrández, An efficient and expandable hardware implementation of multilayer cellular neural networks, Neurocomputing 114 (2013) 54–62.

[125] S. Chakradhar, M. Sankaradas, V. Jakkula, S. Cadambi, A dynamically configurable coprocessor for convolutional neural networks, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2010.

[126] Y. Wang, H. Li, X. Li, Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016, p. 13.

[127] C. Zhang, Z. Fang, P. Zhou, P. Pan, J. Cong, Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016, pp. 1–8.

[128] A. Rahman, J. Lee, K. Choi, Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array,

in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2016, pp. 1393–1398.

[129] M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in: IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12.

[130] M. Gao, J. Pu, X. Yang, M. Horowitz, C. Kozyrakis, Tetris: Scalable and efficient neural network acceleration with 3d memory, in: ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 751–764.

[131] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, Y. Chen, Dadiannao: A neural network supercomputer, IEEE Transactions on Computers 66 (1) (2017) 73–88.

[132] S. Wang, D. Zhou, X. Han, T. Yoshimura, Chain-NN: An energy-efficient 1D chain architecture for accelerating deep convolutional neural networks, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2017, pp. 1032–1037.

[133] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, E. Culurciello, Hardware accelerated convolutional neural networks for synthetic vision systems, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2010, pp. 257–260.

[134] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, Y. Chen, Cambricon-X: An accelerator for sparse neural networks, in: IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12.

[135] H. Kwon, A. Samajdar, T. Krishna, MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects, in: ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2018, pp. 461–475.

[136] T. Gokmen, Y. Vlasov, Acceleration of deep neural network training with resistive cross-point devices: Design considerations, Frontiers in Neuroscience 10.

[137] B. L. Jackson, B. Rajendran, G. S. Corrado, M. Breitwisch, G. W. Burr, R. Cheek, K. Gopalakrishnan, S. Raoux, C. T. Rettner, A. Padilla, et al., Nanoscale electronic synapses using phase change devices, ACM Journal on Emerging Technologies in Computing Systems (JETC) 9 (2) (2013) 12.

[138] S. Saïghi, C. G. Mayr, T. Serrano Gotarredona, H. Schmidt, G. Lecerf, J. Tomas, J. Grollier, S. Boyn, A. F. Vincent, D. Querlioz, et al., Plasticity in memristive devices for spiking neural networks, Frontiers in Neuroscience 9 (2015) 51.

[139] J. Seo, B. Lin, M. Kim, P. Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, S. Vrudhula, S. Yu, J. Ye, et al., On-chip sparse learning acceleration with CMOS and resistive synaptic devices, IEEE Transactions on Nanotechnology (TNANO) 14 (6) (2015) 969–979.

[140] X. Zeng, S. Wen, Z. Zeng, T. Huang, Design of memristor-based image convolution calculation in convolutional neural network, Neural Computing and Applications (2016) 1–6.

[141] Y. Shim, A. Sengupta, K. Roy, Low-power approximate convolution computing unit with domain-wall motion based spin-memristor for image processing applications, in: ACM/IEEE Design Automation Conference (DAC), 2016, pp. 1–6.

[142] L. Ni, H. Huang, H. Yu, On-line machine learning accelerator on digital RRAM-crossbar, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 113–116.

[143] Z. Xu, A. Mohanty, P. Y. Chen, D. Kadetotad, B. Lin, J. Ye, S. Vrudhula, S. Yu, J. Seo, Y. Cao, Parallel programming of resistive cross-point array for synaptic plasticity, Procedia Computer Science 41 (2014) 126–133.

[144] M. Prezioso, F. Merrikh Bayat, B. Hoskins, G. Adam, K. K. Likharev, D. B. Strukov, Training and operation of an integrated neuromorphic network based on metal-oxide memristors, Nature 521 (7550) (2015) 61–64.

[145] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, H. Yang, TIME: A training-in-memory architecture for memristor-based deep neural networks, in: ACM/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

[146] L. Song, Y. Wang, Y. Han, H. Li, Y. Cheng, X. Li, Stt-ram buffer design for precision-tunable general-purpose neural network accelerator, IEEE Transactions on Very Large Scale Integration Systems (TVLSI) 25 (4) (2017) 1285–1296.

[147] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, R. S. Williams, Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication, in: ACM/IEEE Design Automation Conference (DAC), 2016, p. 19.

[148] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, H. Yang, Switched by input: Power efficient structure for RRAM-based convolutional neural network, in: ACM/IEEE Design Automation Conference (DAC), 2016, pp. 1–6.

[149] A. Ankit, A. Sengupta, P. Panda, K. Roy, Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks, in: ACM/IEEE Design Automation Conference (DAC), 2017, pp. 27:1–27:6.

[150] B. Liu, M. Wang, H. Foroosh, M. Tappen, M. Pensky, Sparse convolutional neural networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 806–814.

[151] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Conference on Neural Information Processing Systems (NIPS), 2015, pp. 1135–1143.

[152] M. Courbariaux, I. Hubara, D. Soudry, R. El Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1, arXiv preprint arXiv:1602.02830.

[153] S. S. Liew, M. Khalil Hani, R. Bakhteri, An optimized second order stochastic learning algorithm for neural network training, Neurocomputing 186 (2016) 74–89.

[154] H. Cho, M. K. An, Co-clustering algorithm: Batch, mini-batch, and online, International Journal of Information and Electronics Engineering 4 (5) (2014) 340.

[155] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al., Large scale distributed deep networks, in: Conference on Neural Information Processing Systems (NIPS), 2012, pp. 1223–1231.

[156] Q. Meng, W. Chen, J. Yu, T. Wang, Z. M. Ma, T.-Y. Liu, Asynchronous accelerated stochastic gradient descent, in: International Joint Conference on Artificial Intelligence (IJCAI), 2016.

[157] O. Rippel, J. Snoek, R. P. Adams, Spectral representations for convolutional neural networks, in: Conference on Neural Information Processing Systems (NIPS), 2015, pp. 2449–2457.

[158] Z. Du, S. Liu, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, Q. Guo, X. Feng, Y. Chen, et al., An accelerator for high efficient vision processing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 36 (2) (2017) 227–240.

[159] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, W. J. Dally, Eie: efficient inference engine on compressed deep neural network, in: IEEE/ACM International Symposium on Computer Architecture (ISCA), 2016, pp. 243–254.

[160] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, et al., Going deeper with embedded FPGA platform for convolutional neural network, in: ACM Sympo-

sium on FPGAs, 2016, pp. 26–35.

[161] B. Moons, M. Verhelst, An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS, IEEE Journal Solid-State Circuits 52 (4) (2017) 903–914.

[162] Y. H. Chen, T. Krishna, J. S. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, IEEE Journal Solid-State Circuits 52 (1) (2017) 127–138.