Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang

Model Compression and Acceleration for Deep Neural Networks

The principles, progress, and challenges

Digital Object Identifier 10.1109/MSP.2017.2765695
Date of publication: 9 January 2018

In recent years, deep neural networks (DNNs) have received increased attention, have been applied to different applications, and achieved dramatic accuracy improvements in many tasks. These works rely on deep networks with millions or even billions of parameters, and the availability of graphics processing units (GPUs) with very high computation capability plays a key role in their success. For example, Krizhevsky et al. [1] achieved breakthrough results in the 2012 ImageNet Challenge using a network containing 60 million parameters with five convolutional layers and three fully connected layers. Usually, it takes two to three days to train the whole model on the ImagetNet data set with an NVIDIA K40 machine. In another example, the top face-verification results from the Labeled Faces in the Wild (LFW) data set were obtained with networks containing hundreds of millions of parameters, using a mix of convolutional, locally connected, and fully connected layers [2], [3]. It is also very time-consuming to train such a model to obtain a reasonable performance. In architectures that only rely on fully connected layers, the number of parameters can grow to billions [4].

Introduction

As larger neural networks with more layers and nodes are considered, reducing their storage and computational cost becomes critical, especially for some real-time applications such as online learning and incremental learning. In addition, recent years witnessed significant progress in virtual reality, augmented reality, and smart wearable devices, creating unprecedented opportunities for researchers to tackle fundamental challenges in deploying deep-learning systems to portable devices with limited resources [e.g., memory, central processing units (CPUs), energy, bandwidth]. Efficient deep-learning methods can have a significant impact on distributed systems, embedded devices, and field-programmable gate array (FPGA) for artificial intelligence (AI). For example, the residual network-50 (ResNet-50) [5], which has 50 convolutional layers, needs more than 95 megabytes of memory for storage, and numerous floating number multiplications for

calculating each image. After discarding some redundant weights, the network still works as usual but saved more than 75% of parameters and 50% computational time. For devices like cell phones and FPGAs with only several megabyte resources, how to compact the models used on them is also important.

Achieving these goals calls for joint solutions from many disciplines, including but not limited to machine learning, optimization, computer architecture, data compression, indexing, and hardware design. In this article, we review recent works on compressing and accelerating DNNs, which attracted much attention from the deep-learning community and has already achieved significant progress in past years.

As larger neural networks with more layers and nodes are considered, reducing their storage and computational cost becomes critical, especially for some real-time applications such as online learning and incremental learning.

We classify these approaches into four categories:

1) *Parameter pruning and sharing*: The parameter pruning and sharing-based methods explore the redundancy in the model parameters and try to remove the redundant and noncritical ones.
2) *Low-rank factorizatio*n: Low-rank factorization-based techniques use matrix/tensor decomposition to estimate the informative parameters of the deep convolutional neural networks (CNNs).
3) *Transferred/compact convolutional filters*: The transferred/compact convolutional filters-based approaches design special structural convolutional filters to reduce the storage and computation complexity.
4) *Knowledge distillation* (*KD*): The KD methods learn a distilled model and train a more compact neural network to reproduce the output of a larger network.

In Table 1, we briefly summarize these four types of methods. Generally, the parameter pruning and sharing, low-rank factorization, and KD approaches can be used in DNNs with fully connected layers and convolutional layers, achieving comparable performances. On the other hand, methods using transferred/compact filters are designed for models with convolutional layers only. Low-rank factorization and transferred/compact filters-based approaches provide an end-to-end pipeline and can be easily implemented in a CPU/GPU environment, which is straightforward, while parameter pruning and sharing use different methods such as vector quantization, binary coding, and sparse constraints to perform the task. Usually, it will take several steps to achieve the goal.

Regarding training protocols, models based on parameter pruning/sharing low-rank factorization can be extracted from pretrained ones or trained from scratch, while the transferred/compact filter and KD models can only support training from scratch. These methods are independently designed and complement each other. For example, transferred layers and parameter pruning and sharing can be used together, and model quantization and binarization can be used together with low-rank approximations to achieve further speedup. We will describe the details of each theme and their properties, strengths, and drawbacks in the following sections.

Parameter pruning and sharing

An early work that showed that network pruning is effective in reducing the network complexity and addressed the overfitting problem is [6]. Since then, it has been widely studied to compress DNN models, trying to remove parameters that are not crucial to the model performance. These techniques can be further classified into three categories: model quantization and binarization, parameter sharing, and structural matrix.

Quantization and binarization

Network quantization compresses the original network by reducing the number of bits required to represent each weight. Gong et al. [6] and Wu et al. [7] applied k-means scalar quantization to the parameter values. Vanhoucke et al. [8] showed that 8-bit quantization of the parameters can result in significant speedup with minimal loss of accuracy. The work in [9] used

Table 1. A summary of different approaches for network compression.

Theme Name	Description	Applications	More Details
Parameter pruning and sharing	Reducing redundant parameters that are not sensitive to the performance	Convolutional layer and fully connected layer	Robust to various settings, can achieve good performance, can support both training from scratch and pretrained model
Low-rank factorization	Using matrix/tensor decomposition to estimate the informative parameters	Convolutional layer and fully connected layer	Standardized pipeline, easily implemented, can support both training from scratch and pretrained model
Transferred/compact convolutional filters	Designing special structural convolutional filters to save parameters	Only for convolutional layer	Algorithms are dependent on applications, usually achieve good performance, only support training from scratch
KD	Training a compact neural network with distilled knowledge of a large model	Convolutional layer and fully connected layer	Model performances are sensitive to applications and network structure, only support training from scratch

16-bit fixed-point representation in stochastic rounding-based CNN training, which significantly reduced memory usage and float- point operations with little loss in classification accuracy.

The method proposed in [10] first pruned the unimportant connections and retrained the sparsely connected networks. Then it quantized the link weights using weight-sharing, and then applied Huffman coding to the quantized weights as well as the codebook to further reduce the rate. As shown in Figure 1, it starts by learning the connectivity via normal network training, followed by pruning the small-weight connections. Finally, the network is retrained to learn the final weights for the remaining sparse connections. This work achieves the state-of-the-art performance among all parameter quantization-based methods. It was shown in [11] that Hessian weight could be used to measure the importance of network parameters and proposed to minimize Hessian-weighted quantization errors on average for clustering network parameters. A novel quantization framework was introduced in [12], which reduced the precision of network weights to ternary values.

Network pruning and sharing has been used both to reduce network complexity and to address the overfitting issue.

In the extreme case of 1-bit representation of each weight, i.e., binary weight neural networks, there are also many works that directly train CNNs with binary weights; for instance, BinaryConnect [13], BinaryNet [14], and XNORNetworks [15]. The main idea is to directly learn binary weights or activations during the model training. The systematic study in [16] showed that networks trained with backpropagation could be robust against (robust against or resilient to) specific weight distortions, including binary weights.

Drawbacks

However, the accuracy of such binary nets is significantly lowered when dealing with large CNNs such as GoogleNet. Another drawback of these binary nets is that existing binarization schemes are based on simple matrix approximations and ignore the effect of binarization on the accuracy loss. To address this issue, the work in [17] proposed a proximal Newton algorithm with diagonal Hessian approximation that directly minimizes the loss with respect to the binary weights. The work in [18] significantly reduced the time on float-point multiplication in the training stage by stochastically binarizing weights and converting multiplications in the hidden state computation to sign changes.

Pruning and sharing

Network pruning and sharing has been used both to reduce network complexity and to address the overfitting issue. An early approach to pruning was biased weight decay [19]. The optimal brain damage [20] and the optimal brain surgeon [21] methods reduced the number of connections based on the Hessian of the loss function, and their works suggested that such pruning gave higher accuracy than magnitude-based pruning such as the weight decay method. Those methods supported training from scratch.

A recent trend in this direction is to prune redundant, non-informative weights in a pretrained CNN model. For example, Srinivas and Babu [22] explored the redundancy among neurons and proposed a data-free pruning method to remove redundant neurons. Han et al. [23] proposed to reduce the total number of parameters and operations in the entire network. Chen et al. [24] proposed a HashedNets model that used a low-cost hash function to group weights into hash buckets for parameter sharing. The deep compression method in [10] removed the redundant connections and quantized the weights and then used Huffman coding to encode the quantized weights. In [25], a simple regularization method based on soft weight-sharing was proposed, which

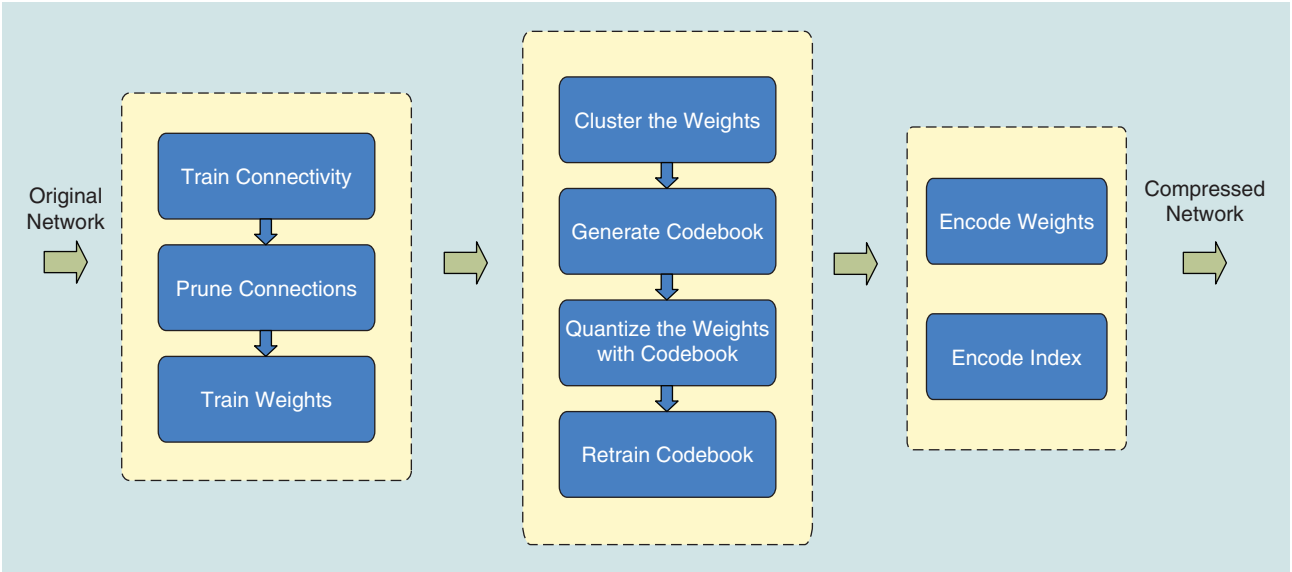

FIGURE 1. The three-stage compression method proposed in [10]: pruning, quantization, and encoding. The input is the original model, and the output is the compression model.

included both quantization and pruning in one simple (re)training procedure. It is worth noting that the aforementioned pruning schemes typically produce connection pruning in CNNs.

There is also growing interest in training compact CNNs with sparsity constraints. Those sparsity constraints are typically introduced in the optimization problem as l_0 or l_1-norm regularizers. The work in [26] imposed group sparsity constraints on the convolutional filters to achieve structured brain damage, i.e., pruning entries of the convolution kernels in a group-wise fashion. In [27], a group-sparse regularizer on neurons was introduced during the training stage to learn compact CNNs with reduced filters. Wen et al. [28] added a structured sparsity regularizer on each layer to reduce trivial filters, channels, or even layers. In filter-level pruning, all of the aforementioned works used $l_{2,1}$-norm regularizers. The work in [29] used l_1-norm to select and prune unimportant filters.

CNNs are parameter-efficient due to exploring the translation invariant property of the representations to input image, which is the key to the success of training very deep models without severe overfitting.

Drawbacks

There are some potential issues of the pruning and sharing works. First, pruning with l_1 or l_2 regularization requires more iterations to converge. Furthermore, all pruning criteria require manual setup of sensitivity for layers, which demands fine-tuning of the parameters and could be cumbersome for some applications.

Designing the structural matrix

In architectures that contain only fully connected layers, the number of parameters can grow up to billions [4]. Thus, it is critical to explore this redundancy of parameters in fully connected layers, which is often the bottleneck in terms of memory consumption. These network layers use the nonlinear transforms $f(\mathbf{x}, \mathbf{M}) = \sigma(\mathbf{Mx})$, where $\sigma(\cdot)$ is an element-wise nonlinear operator, $\mathbf{x}$ is the input vector, and $\mathbf{M}$ is the $m \times n$ matrix of parameters. When $\mathbf{M}$ is a large general dense matrix, the cost of storing mn parameters and computing matrix-vector products in $O(mn)$ time. Thus, an intuitive way to prune parameters is to impose $\mathbf{x}$ as a parameterized structural matrix. An $m \times n$ matrix that can be described using much fewer parameters than mn is called a *structured* matrix. Typically, the structure should not only reduce the memory cost but also dramatically accelerate the inference and training stage via fast matrix-vector multiplication and gradient computations.

Following this direction, the work in [30] proposed a simple and efficient approach based on circulant projections, while maintaining competitive error rates. Given a vector $\mathbf{r} = (r_0, r_1, \ldots, r_{d-1})$, a circulant matrix $\mathbf{R} \in \mathbf{R}^{d \times d}$ is defined as

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

Thus the memory cost becomes $O(d)$ instead of $O(d^2)$. This circulant structure also enables the use of fast Fourier transform (FFT) to speed up the computation. Given a d-dimensional vector $\mathbf{r}$, the 1-layer circulant neural network in (1) has time complexity of $O(d \log d)$.

In [31], a novel adaptive fastfood transform was introduced to reparameterize the matrix-vector multiplication of fully connected layers. The adaptive fastfood transform matrix $\mathbf{R} \in \mathbf{R}^{n \times d}$ was defined as

$$\mathbf{R} = \mathbf{SHG\Pi HB}. \quad (2)$$

Here, $\mathbf{S}, \mathbf{G}$, and $\mathbf{B}$ are random diagonal matrices. $\mathbf{\Pi} \in \{0,1\}^{d \times d}$ is a random permutation matrix and $\mathbf{H}$ denotes the Walsh–Hadamard matrix. Reparameterizing a fully connected layer with d inputs and n outputs using the adaptive fastfood transform reduces the storage and the computational costs from $O(nd)$ to $O(n)$ and from $O(nd)$ to $O(n \log d)$, respectively.

The work in [32] showed the effectiveness of the new notion of parsimony in the theory of structured matrices. Their proposed method can be extended to various other structured matrix classes, including block and multilevel Toeplitz-like [33] matrices related to multidimensional convolution [34].

Drawbacks

One potential problem of this kind of approach is that the structural constraint will cause loss in accuracy since the constraint might bring bias to the model. On the other hand, how to find a proper structural matrix is difficult. There is no theoretical way from which to derive it.

Low-rank factorization and sparsity

As convolution operations constitute the bulk of all computations in CNNs, simplifying the convolution layer would have a direct impact on the overall speedup. The convolution kernels in a typical CNN is a four-dimensional tensor. The key observation is that there might be a significant amount of redundancy in the tensor. Ideas based on tensor decomposition seem to be a particularly promising way to remove the redundancy. Regarding to the fully connected layer, it can be viewed as a two-dimensional (2-D) matrix and the low-rankness can also help.

Using low-rank filters to accelerate convolution has a long history. Typical examples include high-dimensional discrete cosine transform (DCT) and wavelet systems constructed from one-dimensional (1-D) DCT transform and 1-D wavelets, respectively, using tensor products. In the context of dictionary learning, Rigamonti et al. [35] suggested learning separable 1-D filters. In [36], a few low-rank approximation and clustering schemes for the convolutional kernels were proposed. They achieved 2× speedup for a single convolutional layer with 1% drop in classification accuracy. The work in [37] suggested using different tensor decomposition schemes, reporting a 4.5× speedup with 1% drop in accuracy

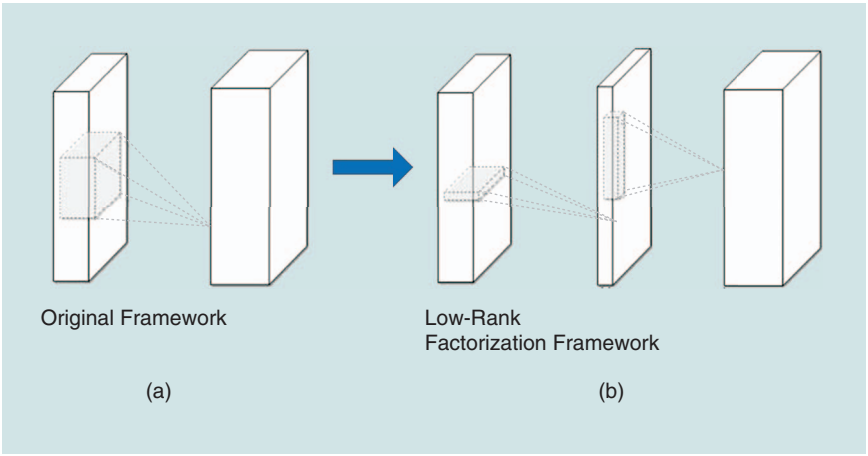

FIGURE 2. A typical framework of the low-rank regularization method. (a) is the original convolutional layer, and (b) is the low-rank constraint convolutional layer with rank-K.

in text recognition. In both works, the approximation was done layer by layer. After one layer was approximated by the low-rank filters, the parameters of that layer were fixed, and the layers above were fine-tuned based on a reconstruction error criterion. These are typical low-rank methods for compressing 2-D convolutional layers, which is described in Figure 2. In [38], canonical polyadic (CP) decomposition of the kernel tensors was proposed. Their work used nonlinear least squares to compute the CP decomposition, which was also based on the tensor decomposition idea. In [39], a new algorithm for computing the low-rank tensor decomposition and a new method for training low-rank constrained CNNs from scratch were proposed. It used batch normalization (BN) to transform the activations of the internal hidden units, and it was shown to be an effective way to deal with the exploding or vanishing gradients.

In principle, both the CP decomposition scheme and the decomposition scheme in [39] (BN low-rank) can be used to train CNNs from scratch. For the CP decomposition, finding the best low-rank approximation is an ill-posed problem, and the best rank-K approximation may not exist in the general case. For the scheme in [39], the decomposition always exists and can achieve better performance than general CP. Table 2 lists a performance comparison of both methods. The actual speedup and compression rates are used to measure the performances. We can see that the BN version can achieve slightly better performance while the CP version gives higher compression rates.

Note that the fully connected layers can be viewed as a 2-D matrix and thus the aforementioned methods can also be applied there. There are several classical works on exploiting low-rankness in fully connected layers. For instance, Misha et al. [40] reduced the number of dynamic parameters in deep models using the low-rank method. Reference [41] explored a low-rank matrix factorization of the final weight layer in a DNN for acoustic modeling.

Table 2. Comparisons between the low-rank models and their baselines on ILSVRC-2012.

Model	TOP-5 Accuracy	Speedup	Compression Rate
AlexNet	80.03%	1	1
BN low-rank	80.56%	1.09	4.94
CP low-rank	79.66%	1.82	5
VGG-16	90.60%	1	1
BN low-rank	90.47%	1.53	2.72
CP low-rank	90.31%	2.05	2.75
GoogleNet	92.21%	1	1
BN low-rank	91.88%	1.08	2.79
CP low-rank	91.79%	1.20	2.84

Drawbacks

Low-rank approaches are straightforward for model compression and acceleration. The idea complements recent advances in deep learning such as dropout, rectified units, and maxout. However, the implementation is not that easy since it involves a decomposition operation, which is computationally expensive. Another issue is that current methods perform low-rank approximation layer by layer, and thus cannot perform global parameter compression, which is important as different layers hold different information. Finally, factorization requires extensive model retraining to achieve convergence when compared to the original model.

Transferred/compact convolutional filters

CNNs are parameter-efficient due to exploring the translation invariant property of the representations to input image, which is the key to the success of training very deep models without severe overfitting. Although a strong theory is currently missing, a large amount of empirical evidence supports the notion that both the translation invariant property and convolutional weight-sharing are important for good predictive performance. The idea of using transferred convolutional filters to compress CNN models is motivated by recent works in [42], which introduced the equivariant group theory. Let $\mathbf{x}$ be an input, $\Phi(\cdot)$ be a network or layer, and $\mathcal{T}(\cdot)$ be the transform matrix. The concept of equivariance is defined as

$$\mathcal{T}'\Phi(\mathbf{x}) = \Phi(\mathcal{T}\mathbf{x}), \tag{3}$$

which says that transforming the input $\mathbf{x}$ by the transform $\mathcal{T}(\cdot)$ and then passing it through the network or layer $\Phi(\cdot)$ should give the same result as first mapping $\mathbf{x}$ through the network and then transforming the representation. Note that,

in [42], the transforms $\mathcal{T}(\cdot)$ and $\mathcal{T}'(\cdot)$ are not necessarily the same as they operate on different objects. According to this theory, it is reasonable to apply the transform to layers or filters $\Phi(\cdot)$ to compress the whole network models. From empirical observation, deep CNNs also benefit from using a large set of convolutional filters by applying a certain transform $\mathcal{T}(\cdot)$ to a small set of base filters since it acts as a regularizer for the model.

Following this trend, there are many recent works proposed to build a convolutional layer from a set of base filters [42]–[45]. What they have in common is that the transform $\mathcal{T}(\cdot)$ lies in the family of functions that only operate in the spatial domain of the convolutional filters. For example, the work in [44] found that the lower convolution layers of CNNs learned redundant filters to extract both positive and negative phase information of an input signal, and defined $\mathcal{T}(\cdot)$ to be the simple negation function

$$\mathcal{T}(\mathbf{W}_x) = \mathbf{W}_x^{-}. \tag{4}$$

Here, $\mathbf{W}_x$ is the basis convolutional filter and $\mathbf{W}_x^{-}$ is the filter consisting of the shifts whose activation is opposite to that of $\mathbf{W}_x$ and selected after max-pooling operation. By doing this, the work in [44] can easily achieve 2× compression rate on all the convolutional layers. It is also shown that the negation transform acts as a strong regularizer to improve the classification accuracy. The intuition is that the learning algorithm with pair-wise positive-negative constraint can lead to useful convolutional filters instead of redundant ones.

In [45], it was observed that magnitudes of the responses from convolutional kernels had a wide diversity of pattern representations in the network, and it was not proper to discard weaker signals with a single threshold. Thus, a multibias nonlinearity activation function was proposed to generate more patterns in the feature space at low computational cost. The transform $\mathcal{T}(\cdot)$ was define as

$$\mathcal{T}'\Phi(\mathbf{x}) = \mathbf{W}_x + \delta, \tag{5}$$

where δ were the multibias factors. The work in [46] considered a combination of rotation by a multiple of 90° and horizontal/vertical flipping with

$$\mathcal{T}'\Phi(\mathbf{x}) = \mathbf{W}^{T_\theta}, \tag{6}$$

where $\mathbf{W}^{T_\theta}$ was the transformation matrix that rotated the original filters with angle $\theta \in \{90, 180, 270\}$. In [42], the transform was generalized to any angle learned from data, and θ was directly obtained from data. Both [46] and [42] can achieve good classification performance.

Reference [43] defined $\mathcal{T}(\cdot)$ as the set of translation functions applied to 2-D filters

$$\mathcal{T}'\Phi(\mathbf{x}) = T(\cdot, x, y)_{x,\, y \in \{-k,\ldots,k\},\, (x,\, y) \neq (0,\, 0)}, \tag{7}$$

where $T(\cdot, x, y)$ denoted the translation of the first operand by (x, y) along its spatial dimensions, with proper zero padding at borders to maintain the shape. The proposed framework can be used to 1) improve the classification accuracy as a regularized version of maxout networks and 2) to achieve parameter efficiency by flexibly varying their architectures to compress networks.

Table 3 briefly compares the performance of different methods with transferred convolutional filters, using VGGNet (16 layers) as the baseline model. The results are reported on the CIFAR-10 and CIFAR-100 data sets with top-five error rates. It is observed that they can achieve reduction in parameters with little or no drop in classification accuracy.

The basic idea of KD is to distill knowledge from a large teacher model into a small one by learning the class distributions output by the teacher via softened softmax.

Drawbacks

There are several issues that need to be addressed for approaches that apply transfer information to convolutional filters. First, these methods can achieve competitive performance for wide/flat architectures (like VGGNet) but not narrow/special ones (like GoogleNet and ResNet). Second, the transfer assumptions sometimes are too strong to guide the algorithm, making the results unstable on some data sets.

Using a compact filter for convolution can directly reduce the computation cost. The key idea is to replace the loose and overparametric filters with compact blocks to improve the speed, which significantly accelerate CNNs on several benchmarks. Decomposing 3×3 convolution into two 1×1 convolutions was used in [47], which achieved state-of-the-art acceleration performance on object recognition. SqueezeNet [48] was proposed to replace 3×3 convolution with 1×1 convolution, which created a compact neural network with approximately 50 fewer parameters and comparable accuracy when compared to AlexNet.

KD

To the best of our knowledge, exploiting knowledge transfer to compress model was first proposed by Caruana et al. [49]. They trained a compressed model with pseudo-data labeled by an ensemble of strong classifiers and reproduced the output of the original larger network. However, their work is limited to shallow models. The idea has been recently adopted in [50] as KD to compress deep and wide networks into shallower ones, where

Table 3. Comparisons of different approaches based on transferred convolutional filters on CIFAR-10 and CIFAR-100.

Model	CIFAR-100	CIFAR-10	Compression Rate
VGG-16	34.26%	9.85%	1
MBA [45]	33.66%	9.76%	2
CRELU [44]	34.57%	9.92%	2
CIRC [42]	35.15%	10.23%	4
DCNN [43]	33.57%	9.65%	1.62

the compressed model mimicked the function learned by the complex model. The basic idea of KD is to distill knowledge from a large teacher model into a small one by learning the class distributions output by the teacher via softened softmax.

The work in [51] introduced a KD compression framework, which eased the training of deep networks by following a student-teacher paradigm, in which the student was penalized according to a softened version of the teacher's output. The framework compressed an ensemble of deep networks (teacher) into a student network of similar depth. To do so, the student was trained to predict the output of the teacher, as well as the true classification labels. Despite its simplicity, KD demonstrates promising results in various image classification tasks. The work in [52] aimed to address the network compression problem by taking advantage of depth neural networks. It proposed an approach to train thin and deep networks, called *FitNets*, to compress wide and shallower (but still deep) networks. The method was rooted in KD and extended the idea to allow for thinner and deeper student models. To learn from the intermediate representations of the teacher network, FitNet made the student mimic the full feature maps of the teacher. However, such assumptions are too strict since the capacities of teacher and student may differ greatly. In certain circumstances, FitNet may adversely affect the performance and convergence. All the aforementioned methods are validated on the MNIST, CIFAR-10, CIFAR-100, SVHN, and AFLW benchmark data sets, and simulation results show that these methods match or outperform the teacher's performance, while requiring notably fewer parameters and multiplications.

The standard criteria to measure the quality of model compression and acceleration are the compression and the speedup rates.

There are several extensions along this direction of distillation knowledge. The work in [53] trained a parametric student model to approximate a Monte Carlo teacher. The proposed framework used online training and used DNNs for the student model. Different from previous works, which represented the knowledge using the softened label probabilities, [54] represented the knowledge by using the neurons in the higher hidden layer, which preserved as much information as the label probabilities, but are more compact. The work in [55] accelerated the experimentation process by instantaneously transferring the knowledge from a previous network to each new deeper or wider network. The techniques are based on the concept of function-preserving transformations between neural network specifications. Zagoruyko et al. [56] proposed attention transfer to relax the assumption of FitNet. They transferred the attention maps that are summaries of the full activations.

Drawbacks

KD-based approaches can make deeper models thinner and help significantly reduce the computational cost. However, there are a few disadvantages. One of them is that KD can only be applied to classification tasks with softmax loss function, which hinders its usage. Another drawback is that the model assumptions sometimes are too strict to make the performance competitive with other types of approaches.

Other types of approaches

We first summarize the works utilizing attention-based methods. Note that attention-based systems [57] can reduce computations significantly by learning to selectively focus or "attend to" a few, task-relevant input regions. The work in [57] introduced the dynamic capacity network that combined two types of modules: the small subnetworks with low capacity, and the large ones with high capacity. The low-capacity subnetworks were active on the whole input to first find the task-relevant areas in the input, and then the attention mechanism was used to direct the high-capacity subnetworks to focus on the task-relevant regions in the input. By doing this, the size of the CNN model could be significantly reduced.

Following this direction, the work in [58] introduced the conditional computation idea, which only computes the gradient for some important neurons. It proposed a new type of general-purpose neural network component: a sparsely gated mixture-of-experts (MoE) layer. The MoE consisted of a number of experts, each a simple feed-forward neural network, and a trainable gating network that selected a sparse combination of the experts to process each input. In [59], dynamic DNNs (D2NNs) were introduced, which were a type of feed-forward DNN that selected and executed a subset of D2NN neurons based on the input.

There have been other attempts to reduce the number of parameters of neural networks by replacing the fully connected layer with global average pooling [43], [60]. Network architectures, such as GoogleNet or network in network, can achieve state-of-the-art results on several benchmarks by adopting this idea. However, transfer learning, i.e., reusing features learned on the ImageNet data set and applying them to new tasks, is more difficult with this approach. This problem was noted by Szegedy et al. [60] and motivated them to add a linear layer on top of their networks to enable transfer learning.

The work in [61] targeted the ResNet-based model with a spatially varying computation time, called *stochastic depth*, which enabled the seemingly contradictory setup to train short networks and used deep networks at test time. It started with very deep networks and, while during training, for each mini-batch, randomly dropped a subset of layers and bypassed them with the identity function. This model is end-to-end trainable, deterministic, and can be viewed as a black-box feature extractor. Following this direction, the work in [62] proposed a pyramidal residual network with stochastic depth.

Other approaches to reduce the convolutional overheads include using FFT-based convolutions [63] and fast convolution using the Winograd algorithm [64]. Those works only aim to speedup the computation but not reduce the memory storage.

Benchmarks, evaluation, and databases

In the past five years, the deep-learning community has made great efforts in benchmark models. One of the most well-known models used in compression and acceleration for CNNs

is Alexnet [1], which occasionally has been used for assessing the performance of compression. Other popular standard models include LeNets [65], All-CNN-nets [66], and many others. LeNet-300-100 is a fully connected network with two hidden layers, with 300 and 100 neurons each. LeNet-5 is a convolutional network that has two convolutional layers and two fully connected layers. Recently, more state-of-the-art architectures are used as baseline models in many works, including network in networks [67], VGGNets [68], and ResNets [69]. Table 4 summarizes the baseline models commonly used in several typical compression methods.

Proposing some general/unified approaches is one direction that can be taken regarding the use of CNNs in small platforms.

The standard criteria to measure the quality of model compression and acceleration are the compression and the speedup rates. Assume that a is the number of the parameters in the original model M and a^* is that of the compressed model M^*, then the compression rate $\alpha(M, M^*)$ of M^* over M is

$$\alpha(M, M^*) = \frac{a}{a^*}. \tag{8}$$

Another widely used measurement is the index space saving defined in several papers [70], [71] as

$$\beta(M, M^*) = \frac{a - a^*}{a^*}, \tag{9}$$

where a and a are the number of the dimension of the index space in the original model and that of the compressed model, respectively.

Similarly, given the running time s of M and s^* of M^*, the speedup rate $\delta(M, M^*)$ is defined as

$$\delta(M, M^*) = \frac{s}{s^*}. \tag{10}$$

Most work used the average training time per epoch to measure the running time, while in [70] and [71], the average testing time was used. Generally, the compression rate and speedup rate are highly correlated, as smaller models often results in faster computation for both the training and the testing stages.

Good compression methods are expected to achieve almost the same performance as the original model with much smaller parameters and less computational time. However, for different applications with varying CNN designs, the correlation between parameter size and computational time may be different. For example, it is observed that, for deep CNNs with fully connected layers, most of the parameters are in the fully connected layers; while for image classification tasks, float-point operations are mainly in the first few convolutional layers since each filter is convolved with the whole image, which is usually very large at the beginning. Different applications should focus on different layers.

Discussion and challenges

In this article, we summarized recent works on compressing and accelerating DNNs. Here we discuss more details about how to choose different compression approaches and possible challenges/solutions in this area.

General suggestions

There is no golden rule to measure which one of the four kinds of approaches is the best. How to choose the proper approaches is really dependent on the applications and requirements. Here, we provide some general suggestions.

- If the applications needs compacted models from pretrained models, one can choose either pruning and sharing or low-rank factorization-based methods. If end-to-end solutions are needed for the problem, the low-rank and transferred convolutional filters approaches are preferred.
- For applications in some specific domains, methods with human prior (like the transferred convolutional filters and structural matrix) sometimes have benefits. For example, when conducting medical images classification, transferred convolutional filters should work well as medical images (like organs) do have the rotation transformation property.
- Usually, the approaches of pruning and sharing could give a reasonable compression rate while not hurting the accuracy. Thus, for applications that require stable model accuracy, it is better to utilize pruning and sharing.
- If a problem involves small- or medium-size data sets, one can try the KD approaches. The compressed student model can take the benefit of transferring knowledge from the teacher model, making it a robust data set that is not large.
- As we mentioned in the "Introduction," techniques of the four themes are orthogonal. It makes sense to combine two or three of them to maximize the compression/speedup rates. For some specific applications, like object detection, which requires both convolutional and fully connected layers, one can compress the convolutional layers with low-rank factorization and the fully connected layers with a pruning method.

Table 4. A summary of baseline models used in different representative works of network compression.

Baseline Models	Representative Works
Alexnet [1]	Structural matrix [30]–[32]
	Low-rank factorization [39]
Network in network [67]	Low-rank factorization [39]
VGGNets [68]	Transferred filters [43]
	Low-rank factorization [39]
ResNets [69]	Compact filters [48], stochastic depth [61]
	Parameter sharing [25]
All-CNN-nets [66]	Transferred filters [44]
LeNets [65]	Parameter sharing [25]
	Parameter pruning [21], [23]

Technique challenges

Techniques for deep model compression and acceleration are still in the early stages, and the following challenges still need to be addressed.

- Most of the current state-of-the-art approaches are built on well-designed CNN models, which have limited freedom to change the configuration (e.g., network structural, hyperparameters). To handle more complicated tasks, it should provide more plausible ways to configure the compressed models.
- Pruning is an effective way to compress and accelerate CNNs. Current pruning techniques are mostly designed to eliminate connections between neurons. On the other hand, a pruning channel can directly reduce the feature map width and shrink the model into a thinner one. It is efficient but also challenging because removing channels might dramatically change the input of the following layer. It is important to focus on how to address this issue.
- As we mentioned previously, methods of structural matrix and transferred convolutional filters impose prior human knowledge to the model, which could significantly affect the performance and stability. It is critical to investigate how to control the impact of the imposed prior knowledge.
- The methods of KD provide many benefits such as directly accelerating the model without special hardware or implementations. It is still worth it to develop KD-based approaches and explore how to improve the performance.
- Hardware constraints in various of small platforms (e.g., mobile, robotic, self-driving cars) are still a major problem that hinder the extension of deep CNNs. How to make full use of the limited computational source available and how to design special compression methods for such platforms are still challenges that need to be addressed.

Good compression methods are expected to achieve almost the same performance as the original model with much smaller parameters and less computational time.

Possible solutions

To solve the hyperparameters configuration problem, we can rely on the recent learning-to-learn strategy [72], [73]. This framework provides a mechanism, allowing the algorithm to automatically learn how to exploit structure in the problem of interest. There are two different ways to combine the learning-to-learn module with the model compression. The first designs compression and learning-to-learn simultaneously, while the second way first configures the model with learn-to-learning and then prunes the parameters.

Channel pruning provides the efficiency benefit on both CPUs and GPUs because no special implementation is required. But it is also challenging to handle the input configuration. One possible solution is to use the training-based channel pruning methods [74], which focus on imposing sparse constraints on weights during training, and could adaptively determine hyperparameters. However, training from scratch for such a method is costly for very deep CNNs.

Exploring new types of knowledge in the teacher models and transferring it to the student models is useful for the KD approaches. Instead of directly reducing and transferring parameters from the teacher models, passing selectivity knowledge of neurons could be helpful. One can derive a way to select essential neurons related to the task. The intuition is that, if a neuron is activated in certain regions or samples, this implies these regions or samples share some common properties that may relate to the task. Performing such steps is time-consuming, thus efficient implementation is important.

For methods with convolutional filters and the structural matrix, we can conclude that the transformation lies in the family of functions that only operations on the spatial dimensions. Hence, to address the imposed prior issue, one solution is to provide a generalization of the aforementioned approaches in two aspects: 1) instead of limiting the transformation to belong to a set of predefined transformations, let it be the whole family of spatial transformations applied to 2-D filters or the matrix, and 2) learn the transformation jointly with all of the model parameters.

Proposing some general/unified approaches is one direction that can be taken regarding the use of CNNs in small platforms. Yuhen et al. [75] presented a feature map dimensionality reduction method by excavating and removing redundancy in feature maps generated by different filters, which could also preserve intrinsic information of the original network. The idea can be extended to make CNNs more applicable for different platforms. The work in [76] proposed a one-shot whole network compression scheme consisting of three components: rank selection, low-rank tensor decomposition, and fine-tuning to make deep CNNs work in mobile devices. From the systematic side, Facebook released the platform Caffe2 [77], which employed a particularly lightweight and modular framework and included mobile-specific optimizations based on the hardware design. Caffe2 can help developers and researchers train large machine-learning models and deliver AI on mobile devices.

Acknowledgments

We would like to thank the reviewers and broader community for their feedback on this survey. In particular, we would like to thank Hong Zhao from the Department of Automation of Tsinghua University for her help on modifying this article. This research is supported by National Science Foundation of China, grant number 61401169. The corresponding author of this article is Pan Zhou.

Authors

Yu Cheng (chengyu@us.ibm.com) received his bachelor's degree in automation from Tsinghua University, Beijing, China, in 2010 and his Ph.D. degree in computer science from Northwestern University, Evanston, Illinois in 2015. Currently, he is a research staff member at AI Foundations Lab, IBM T.J. Watson Research Center, Yorktown Heights, New York. His research is focused on deep learning in general, with specific interests in deep generative models and deep models

compression. He also has published many works regarding the applications of deep learning in computer vision and natural language processing.

Duo Wang (d-wang15@mails.tsinghua.edu.cn) received the B.S. degree in automation from the Harbin Institute of Technology, China, in 2015, where he is currently pursuing his Ph.D. degree in the Department of Automation, Tsinghua University. His research interests are deep/machine learning and their applications in computer vision and robotics vision.

Pan Zhou (panzhou@hust.edu.cn) received his B.S. degree in the Advanced Class of Huazhong University of Science and Technology (HUST), Wuhan China, and his M.S. degree in electronics and information engineering from the same university in 2006 and 2008, respectively. He received his Ph.D. degree from the School of Electrical and Computer Engineering at the Georgia Institute of Technology, Atlanta in 2011. Currently, he is an associate professor with School of Electronic Information and Communications, HUST. His research interests include big data analytics and machine learning, security and privacy, and information networks.

Tao Zhang (taozhang@mail.tsinghua.edu.cn) received his B.S., M.S., and Ph.D. degrees from Tsinghua University, Beijing, China, in 1993, 1995, and 1999, respectively, and his Ph.D. degree from Saga University, Japan, in 2002, all in control engineering. He is a professor with the Department of Automation, Tsinghua University. His current research interests include artificial intelligence, robotics, image processing, control theory, and control of spacecraft.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Conf. Neural Information Processing Systems*, 2012, pp. 1097–1105.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2014, pp. 1701–1708.

[3] Y. Sun, X. Wang, and X. Tang, "Deeply learned face representations are sparse, selective, and robust," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2015, pp. pp. 2892–2900.

[4] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep networks," in *Proc. Conf. Neural Information Processing Systems*, 2012, pp. 1223–1231.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Computing Res. Repository, vol. abs/1512.03385, 2015. [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf

[6] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," Computing Res. Repository, vol. abs/1412.6115, 2014. [Online]. Available: https://arxiv.org/pdf/1412.6115.pdf

[7] Y. W. Q. H. Jiaxiang Wu, C. Leng, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Computer Vision Pattern Recognition,* 2016, pp. 4820–4828.

[8] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Conf. Neural Information Processing Systems Deep Learning and Unsupervised Feature Learning Workshop*, 2011.

[9] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Machine Learning,* 2015, vol. 37, pp. 1737–1746.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learning Representations*, 2016.

[11] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," Computing Res. Repository, vol. abs/1612.01543, 2016. [Online]. Available: https://arxiv.org/abs/1612.01543

[12] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," arXiv Preprint*,* arXiv:1612.01064, 2016.

[13] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Advances Neural Information Processing Systems Annu. Conf.*, 2015, pp. 3123–3131.

[14] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or −1," Computing Res. Repository*,* vol. abs/1602.02830, 2016. [Online]. Available: https://arxiv.org/abs/1602.02830

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. European Conf. Computer Vision*, 2016, pp. 525–542.

[16] P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," Computing Res. Repository, vol. abs/1606.01981, 2016. [Online]. Available: https://arxiv.org/abs/1606.01981

[17] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," Computing Res. Repository, vol. abs/1611.01600, 2016. [Online]. Available: https://arxiv.org/abs/1611.01600

[18] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," Computing Res. Repository, vol. abs/1510.03009, 2015. [Online]. Available: https://arxiv.org/abs/1510.03009

[19] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," *Adv. Neural Inform. Process. Syst. 1*, 1989, pp. 177–185.

[20] Y. L. Cun, J. S. Denker, and S. A. Solla, "Advances in neural information processing systems 2," in *Optimal Brain Damage*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 598–605.

[21] B. Hassibi, D. G. Stork, and S. C. R. Com, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, vol. 5. San Mateo, CA: Morgan Kaufmann, 1993, pp. 164–171.

[22] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *Proc. British Machine Vision Conf.*, 2015, pp. 31.1–31.12.

[23] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Information Processing Systems*, 2015, pp. 1135–1143.

[24] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Machine Learning Research Workshop Conf.*, 2015, pp. 2285–2294.

[25] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," Computing Res. Repository, vol. abs/1702.04008, 2017. [Online]. Available: https://arxiv.org/abs/1702.04008

[26] V. Lebedev and V. S. Lempitsky, "Fast convnets using group-wise brain damage," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2016, pp. 2554–2564.

[27] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact CNNs," in *Proc. European Conf. Computer Vision*, 2016, pp. 662–677.

[28] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Adv. Neural Inform. Process. Syst.*, vol. 29, pp. 2074–2082, 2016.

[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," Computing Res. Repository, vol. abs/1608.08710, 2016. [Online]. Available: https://arxiv.org/abs/1608.08710

[30] Y. Cheng, F. X. Yu, R. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proc. Int. Conf. Computer Vision,* 2015, pp. 2857–2865.

[31] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *Proc. Int. Conf. Computer Vision*, 2015, pp. 1476–1483.

[32] V. Sindhwani, T. Sainath, and S. Kumar. (2015). Structured transforms for small-footprint deep learning. *Advances in Neural Information Processing Systems*, *28*, pp. 3088–3096. [Online]. Available: http://papers.nips.cc/paper/5869-structured-transforms-for-small-footprint-deep-learning.pdf

[33] J. Chun and T. Kailath, *Generalized Displacement Structure for Block-Toeplitz, Toeplitz-Block, and Toeplitz-Derived Matrices*. Berlin, Germany: Springer, 1991, pp. 215–236.

[34] M. V. Rakhuba and I. V. Oseledets. (2015). Fast multidimensional convolution in low-rank tensor formats via cross approximation. *SIAM J. Sci. Comput.*, *37*(2). [Online]. Available: http://dx.doi.org/10.1137/140958529

[35] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, "Learning separable filters," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2013, pp. 2754–2761.

[36] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *Adv. Neural Inform. Process. Syst.* vol. 27, pp. 1269–1277, 2014.

[37] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. British Machine Vision Conf.*, 2014, pp. 1–13.

[38] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," Computing Res. Repository, vol. abs/1412.6553, 2014. [Online]. Available: https://arxiv.org/abs/1412.6553

[39] C. Tai, T. Xiao, X. Wang, and E. Weinan, "Convolutional neural networks with low-rank regularization," Computing Res. Repository, vol. abs/1511.06067, 2015.

[40] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. D. Freitas. (2013). Predicting parameters in deep learning. *Advances in Neural Information Processing Systems*, *26*, 2148–2156. [Online]. Available: http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips26/1053.pdf

[41] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. IEEE Int. Conf. Acoustics Speech Signal Processing*, 2013, pp. 6655–6659.

[42] T. S. Cohen and M. Welling, "Group equivariant convolutional networks," arXiv Preprint, arXiv:1602.07576, 2016.

[43] S. Zhai, Y. Cheng, and Z. M. Zhang, "Doubly convolutional neural networks," in *Proc. Advances Neural Information Processing Systems*, 2016, pp. 1082–1090.

[44] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," arXiv Preprint, arXiv:1603.05201, 2016.

[45] H. Li, W. Ouyang, and X. Wang, "Multi-bias non-linear activation in deep neural networks," arXiv Preprint, arXiv:1604.00676, 2016.

[46] S. Dieleman, J. D Fauw, and K. Kavukcuoglu, "Exploiting cyclic symmetry in convolutional neural networks," in *Proc. 33rd Int. Conf. Machine Learning*, 2016, vol. 48, pp. 1889–1898.

[47] C. Szegedy, S. Ioffe, and V. Vanhoucke. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning, Computing Res. Repository, vol. abs/1602.07261. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1602.html#SzegedyIV16

[48] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," Computing Res. Repository, vol. abs/1612.01051, 2016. [Online]. Available: https://arxiv.org/abs/1612.01051

[49] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. (2006). Model compression. *Proc. 12th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, pp. 535–541. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150464

[50] J. Ba and R. Caruana, "Do deep nets really need to be deep?" *Adv. Neural Inform. Process. Syst.*, vol. 27, pp. 2654–2662, 2014.

[51] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," Computing Res. Repository, vol. abs/1503.02531, 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[52] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," Computing Res. Repository, vol. abs/1412.6550, 2014. [Online]. Available: https://arxiv.org/abs/1412.6550

[53] A. Korattikara Balan, V. Rathod, K. P. Murphy, and M. Welling. (2015). Bayesian dark knowledge. *Advances in Neural Information Processing Systems*, *28*, 3420–3428. [Online]. Available: http://papers.nips.cc/paper/5965-bayesian-dark-knowledge.pdf

[54] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, "Face model compression by distilling knowledge from neurons," in *Proc. 30th AAAI Conf. Artificial Intelligence*, 2016, pp. 3560–3566.

[55] T. Chen, I. J. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *Computing Res. Repository*, vol. abs/1511.05641, 2015. [Online]. Available: https://arxiv.org/abs/1511.05641

[56] S. Zagoruyko and N. Komodakis. (2016). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, *Computing Res. Repository*, vol. abs/1612.03928. [Online]. Available: http://arxiv.org/abs/1612.03928

[57] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. C. Courville, "Dynamic capacity networks," in *Proc. 33rd Int. Conf. Machine Learning*, 2016, pp. 2549–2558.

[58] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. [Online]. Available: https://openreview.net/pdf?id=B1ckMDqlg

[59] D. Wu, L. Pigou, P. Kindermans, N. D. Le, L. Shao, J. Dambre, and J. Odobez, "Deep dynamic neural networks for multimodal gesture segmentation and recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 8, pp. 1583–1597, 2016.

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. (2015). Going deeper with convolutions. *Proc. IEEE Computer Vision Pattern Recognition*. [Online]. Available: http://arxiv.org/abs/1409.4842

[61] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," Computing Res. Repository, vol. arXiv:1603.09382, 2016.

[62] Y. Yamada, M. Iwamura, and K. Kise. (2016). Deep pyramidal residual networks with separated stochastic depth, Computing Res. Repository, vol. abs/1612.01230. [Online]. Available: http://arxiv.org/abs/1612.01230

[63] M. Mathieu, M. Henaff, and Y. Lecun, "Fast training of convolutional networks through FFTs," Computing Res. Repository, vol. arXiv:1312.5851, 2014.

[64] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2016, pp. 4013–4021.

[65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, pp. 2278–2324, 1998.

[66] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," Computing Res. Repository, vol. abs/1412.6806, 2014. [Online]. Available: https://arxiv.org/abs/1412.6806

[67] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. Int. Conf. Learning Representations*, 2014. [Online]. Available: https://arxiv.org/abs/1312.4400

[68] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Computing Res. Repository, vol. abs/1409.1556, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv Preprint, arXiv:1512.03385, 2015.

[70] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. N. Choudhary, and S. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proc. IEEE Int. Conf. Computer Vision*, 2015, pp. 2857–2865.

[71] M. Moczulski, M. Denil, J. Appleyard, and N. de Freitas, "ACDC: A structured efficient linear layer," in *Proc. Int. Conf. Learning Representations*, 2016.

[72] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Proc. Neural Information Processing Systems Conf.*, 2016, pp. 3981–3989.

[73] D. Ha, A. Dai, and Q. Le, "Hypernetworks," in *Proc. Int. Conf. Learning Representations*, 2016.

[74] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Proc. Neural Information Processing Systems Conf.*, 2016, pp. 2270–2278.

[75] Y. Wang, C. Xu, C. Xu, and D. Tao, "Beyond filters: Compact feature map for portable deep model," in *Proc. 34th Int. Conf. Machine Learning*, 2017, pp. 3703–3711.

[76] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," Computing Res. Repository, vol. abs/1511.06530, 2015. [Online]. Available: https://arxiv.org/abs/1511.06530

[77] Facebook, Inc. Caffe2: A new lightweight, modular, and scalable deep learning framework. (2016). [Online]. Available: https://caffe2.ai/

SP