BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 6, 2018
DOI: 10.24425/bpas.2018.125927

Speeding-up convolutional neural networks: A survey

V. LEBEDEV[1,2]* and V. LEMPITSKY[1]

[1]Skolkovo Insitute of Science and Technology, Moscow, Russia
[2]Yandex, Moscow, Russia

Abstract. Convolutional neural networks (CNN) have become ubiquitous in computer vision as well as several other domains, but the sheer size of the modern CNNs means that for the majority of practical applications, a significant speed up and compression are often required. Speeding-up CNNs therefore have become a very active area of research with multiple diverse research directions pursued by many groups in academia and industry. In this short survey, we cover several research directions for speeding up CNNs that have become popular recently. Specifically, we cover approaches based on tensor decompositions, weight quantization, weight pruning, and teacher-student approaches. We also review CNN architectures designed for optimal speed and briefly consider automatic architecture search.

Key words: convolutional neural networks, resource-efficient computation, algorithm optimization..

1. Introduction

Convolutional neural networks (CNNs) have become the most important computational architecture in the computer vision domain, where they are used for image classification, segmentation, detection, filtering and image generation tasks. Beyond computer vision, methods based on CNNs thrive in natural language processing [1], bioinformatics [2] and general-purpose reinforcement learning [3]. Although the basic ideas behind CNNs have emerged by the late 1980s [4, 5], successful application of these ideas to natural images was delayed till the introduction of powerful graphics processing units (GPUs) [6, 7], which are almost exclusively used for CNN training these days.

While GPU remains the dominant architecture for training CNNs within the academic community, the industrial applications of CNNs often require their use in lower-end computing architectures with limited computational power, limited memory and limited battery power. Such architectures include smartphones and wearable devices. For robotics and autonomous driving, GPU-type units may be available, but time constraints are often extremely stringent. For all these reasons, speeding up and compressing CNN models has become a very active and diverse area of research.

This short survey covers (a subset of) this rapidly evolving field, focusing on approaches that aim at building CNNs that are fast at inference time. In general, we group these approaches into six groups:

- The approaches using tensor decompositions of the weights or activations.
- Methods relying on low-precision arithmetic and other quantization techniques. This group includes methods that aim to build fully binarized neural networks.
- Approaches that prune the weigths of larger networks to build their smaller and faster equivalents.
- Teacher-student approaches, which train small networks with the aid of bigger ones.
- Efficient architectures design: methods that pursue heuristic-driven search for the smallest possible CNN architecture.
- Methods for automatic architecture search that aim to replace human-suggested heuristics with an algorithm that designs neural networks automatically.

Below, each group is presented within a separate section. We note that factorization, quantization, and pruning approaches can all be grouped into a "gradual" speed-up super-group. Such approaches start with a pretrained CNN and then interleave the transformation of convolutional layers with fine-tuning operations. Each transformation step leads to a speed-up as well as to the drop in accuracy. The subsequent fine-tuning operation recovers part of the accuracy drop. On the other hand, methods aimed at fast architecture design (including those that do this in an automated way) aim to design an architecture that can be trained from scratch. Finally, the teacher-student methods take the middle path, as they usually design the final architecture in a two-step process, which first trains a slow teacher network and then trains a fast student network using the guidance from the teacher.

Several aspects related to CNN acceleration fall outside of the scope of this short survey. In particular, we do not discuss the hardware-dependent implementation details, although they can play a very important role in practice. This topic, among others, is addressed in the concurrent reviews in [8] and [9]. Also outside of our survey is the group of methods that design architectures that adapt to input samples, and in particular can use early-stopping for simple examples. This promising group of methods includes [10–13]. Also, the focus of this survey on

*e-mail: vadim.lebedev@skoltech.ru

Manuscript submitted 2018-05-07, initially accepted for publication 2018-05-30, published in December 2018.

the general-purpose methods that are usually introduced in the context of image classification. At the same time, we note that a lot of work has been put into architectures that are specific to other computer vision tasks, most notably object detection and semantic segmentation. Such approaches also fall outside the scope of this survey.

2. Tensor decompositions

The convolution layers correspond to the bulk of the inference time in modern CNNs. These layers are based on the generalized convolution operation:

$$V(x, y, k) = \sum_{i=0}^{d_x-1} \sum_{j=0}^{d_y-1} \sum_{c=0}^{C-1} W(i, j, s, k) U(x+i, y+j, c), \quad (1)$$

where U denotes the input 3D tensor containing C 2D image maps, V denotes the output 3D tensor containing N 2D image maps, and W is a four-dimensional convolutional tensor (also often referred to as convolutional kernel or weight tensor) containing $d_x \times d_y \times C \times T$ elements. The out-of-bounds indices in the summation of (1) are handled using so-called padding routines, which are out of the scope of this survey.

The generalized convolution is thus defined by its four-dimensional convolutional tensor W. The idea behind tensor decomposition methods is to decompose this tensor into a product of low-dimensional tensors, resulting in several fast convolutions with fewer operations. Decomposition for speeding-up convolutional filters was initially developed out of the scope of CNNs. Here, [14] considered denoising tasks and approximated convolutional filters by a shared set of separable filters, leading to large speedups with minimal loss in denoising accuracy.

The approach based on separable filters was then extended to convolutional layers of CNNs in [15]. The convolutional tensor with square $d \times d$ filters is approximated and replaced by a product of two convolutional tensors with $1 \times d$ and $d \times 1$ filters and K feature maps between them. The parameter K (the decomposition rank) regulates the speed-accuracy trade-off: small K lead to fast but inaccurate models, while large K allow to reproduce original convolution closely but with high computation time. Carefully tuning K for every approximated layer is a crucial part of speeding up the neural network within this algorithm. Their work describes two optimization approaches, depending on whether the objective minimized by the decomposition procedure measures the error of the filter reconstruction or the error of the unit activation reconstruction. Of the two approaches, the latter is more practical as it can be made a part of the CNN fine-tuning process, which optimizes the training loss used to train CNN over all parameters of the network (although end-to-end fine-tuning was not pursued in [15]).

Originally, the separability was enforced on pretrained network only, but [15] also note that low rank filters can be learned in the discriminative manner, i.e. from scratch and at the same time with the rest of the network. This idea was later incorporated into Inception architectures, starting from the second version [16].

Table 1
Per-pixel operation counts for different approximations of convolutional layers, with the focus on CP-decomposition approaches. The number of operations depends on the size d of the square spatial kernel, the number of input channels C and the output channels N, as well as the hyper-parameters specific to approximation methods: the decomposition rank K, and the number of clusters C_1 and N_1 for input and output channels for the clustering methods from [17]

method	operations
full convolution	CNd^2
two-component [37]	$Kd(C+N)$
monochromatic approximation [16]	$CC_1 + Nd^2$
biclustering + svd [16]	$CC_1K_1 + C_1N_1K_1K_2d_2 + NN_1K_2$
biclustering + outer product [16]	$K(CC_1 + C_1N_1d_2 + N_1N)$
cp-decomposition [44]	$K(C + 2d + N)$
cp-decomposition [4]	$K(C + 2d + N)$

Several weight tensor compression methods based on clustering were proposed by [17]. The common idea is to cluster the tensor slices along one or two of the tensor dimensions, to split up the tensor according to cluster boundaries and then to approximate resulting slices by centroid values ("monochromatic approximation"), via the SVD decomposition, or via the canonical polyadic (CP) decomposition obtained by greedy approach (outer product decomposition). The CP decomposition is one of the generalization of the SVD decomposition to higher-order tensors (a review [20] on such decompositions is highly recommended). The splitting reduces decomposition ranks required for good approximation, but this comes at the cost of increasing the number of tensors that need to be approximated. Another shortcoming of this approach is in a rather large number of hyper-parameters, which makes it increasingly hard to tune for optimal performance.

The work [18] presents the algorithm that also applies the CP-decomposition to the convolutional weight tensor. Instead of clustering, the focus of this work is on tuning the performance of CP-decomposition, which is done in two phases. The first phase starts with with better initial approximation obtained by the non-linear least squares algorithm (instead of the greedy one used in [17]). The second stage finetunes the whole network after the decomposition is applied.

Finetuning the model after decomposition is non-trivial, especially if the decomposition is applied to several layers. Even if the single layer is decomposed, numerical instabilities inside the four convolutional layers introduced by CP-decomposition often lead to the explosion of gradients during fine-tuning. The key problem is that CNN block designed to follow the structure of tensor decomposition does not have non-linearities between the convolutions. When multiple layers are decomposed, the fine-tuning can be done either once after all the decompositions are applied, or iteratively, after every single decomposition. The iterative approach was explored by [19], which also argues for tensor power method [21] to be the most appropriate for obtaining initial decomposition. Training from scratch with the architecture closely related to one obtained with CP-decomposition was implemented in [22].

The work [23] suggests an alternative way to speed up neural networks by applying the low-rank assumption to activations rather than to the weight tensor. This assumption splits one convolutional layer in two, while the weights of each of the two new layers are obtained by solving optimization problems. The focus on activations instead of the weights has two advantages: first, the nonlinearities can be taken into account in the optimization problem formulation, and secondly, in case of multiple layers approximations, the activations of the original network can be used as a target. This idea, called the asymmetric reconstruction limits the accumulation of error from layer to layer. Finally, as in the previous approaches, the whole network can be fine-tuned.

All the methods listed above replace single convolutional layer by a block of smaller convolutions. The comparison of these blocks is presented in Fig. 1.

Other higher-order tensor decompositions have been used for CNN speed-up. Tucker decomposition was applied for speeding up and compression of CNNs in [24] and [25]. Tensor Train (TT) decomposition was applied to fully connected layers of convolutional neural networks by [26]. The main focus of that work is compression, not the speed-up, but the achieved compression rates of up to 200 000 times are impressive.

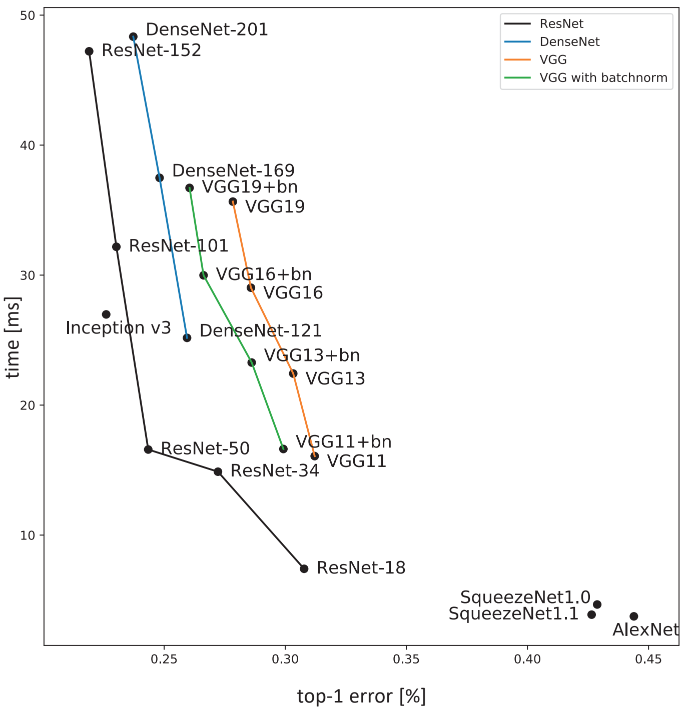

Fig. 2. The trade-off between the inference time on Tesla K40m GPU and the ILSVRC [27] Top-1 classification error for the popular CNN architecutres (PyTorch [28] implementations). Color lines connect groups of similar architectures. Many of the speed-up approaches take some of the charted architectures as starting points

3. Fast architecture design

The research in CNNs has lead to the emergence of several popular families of the architectures. Historically, the search for architectures was driven by the desire to push the classification accuracy (most importantly in the annual ILSVRC [27] challenge), while the inference speed was of a secondary concern. Figure 2 shows runtimes and the ILSVRC accuracies of the resulting architectures. Many of the approaches surveyed below take the architectures reflected in this chart as a starting point for the design process.

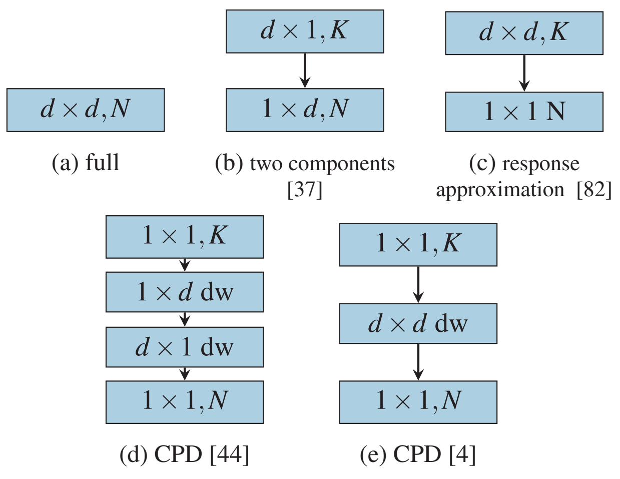

Fig. 1. CNN blocks used by tensor decomposition methods to replicate a single convolutional layer. Each layer here is labeled with its kernel shape and the number of filters. 'dw' stands for depthwise convolution, in which case the number of the channels in the input is the same as on the output

The tensor decomposition approaches are closely related to the task of designing of optimal architectures, which is the topic of this subsection. The methods, described in this section, however, train the designed architectures from scratch. The design choices are often directly influenced by preceeding works on tensor decomposition, such as in the case of [22].

One of the first prominent attempts at building an architecture, which emphasizes efficiency is the Network-In-Network (NIN) architecture proposed in [29]. The basic idea behind NIN is to replace non-linearities within the convolutional network with a more complex function. Multilayer (two-layer) perceptron, which is known to be a universal aproximator, is chosen as a replacement. By sharing the weights of the perceptrons across spatial dimensions one ends up with two 1×1 convolutional layers interleaved with standard non-linearities.

The SqueezeNet [30] is a compact architecture that achieves AlexNet-level performance with $50\times$ less parameters. The ideology behind SqueezeNet is based on three principles:

1. Utilize 1×1 filters instead of 3×3 filters whenever possible. Smaller filters have fewer parameters and need fewer operations.
2. When 3×3 filtering has to be applied, minimize the number of input channels.
3. Downsample late in the network to give most layers chance to work with large high-resolution maps.

The first and the second principles ensure that the model is both small and fast, while the third design principle boosts the accuracy. All three principles combined yield a very small model with a slightly smaller inference time than AlexNet (which serves as the base model).

The MobileNet architecture [31] is prominent among CNN architectures designed for optimal size and inference time. The main idea is to separate filtering and feature construction functions of a convolutional layer into two layers: the depthwise convolution and 1×1 convolution (i.e. a pointwise convolution). This combination is called a depthwise separable convolution and was first popularized in [32].

The MobileNet also employs simple but effective tricks to control architecture performance: the network width (the number of channels) is controlled by the width multipler α, and the input resolution is controlled by the resolution multiplier ρ. In general, changing the network width and the input resolution is a simple way to control trade-offs between the speed and the number of parameters on one hand and the accuracy on the other. Such knob, however, had not been thoroughly investigated in research papers prior to [31].

In comparison with SqueezeNet, MobileNet is capable of achieving higher accuracy with approximately same model size and one tenth of mult-add operations. This advantage in operation number, however, is hard to translate into actual inference speed-up on GPU because depth-wise convolution is not as efficiently implemented on GPU as a regular convolution.

Still, separating the depth-wise and intra-channel convolutions within an architecture has become a popular idea. In modern architectures, this idea leads to 1×1 convolution being dominant in the total computation cost of the model. The only way to squeeze 1×1 convolutions even further is to turn them into *group convolutions* [7], which only mix channels within certain groups of channels. Such grouping, however, would mean that the whole network is divided into thin columns with no connection between them. ShuffleNets [33] address this problem by using the channel shuffle operation. Shuffling of channels between the convolutions allows to enjoy the low costs of group convolutions without splitting the network into disjoint parts. The ShuffleNet architecture uses even smaller number of operations for the same accuracy level compared to MobileNet. Whether this advantage translates to actual timings depends on the efficiency of the available implementations of depthwise and group convolutions.

Yet another variation of MobileNet building block is the EffNet architecture [34], motivated by a careful study. EffNet utilizes depthwise separable convolutions, and pushes it even further by splitting 3×3 convolution into pair of convolutions with 1×3 and 3×1 kernels. The downsampling along one dimension is perfomed with strided convolution, and along other – via 2×1 pooling.

The final logical step in the movement towards smaller filters in CNNs would be a complete rejection of convolutions with filers larger than 1×1. The problem with this kind of CNN is that adjacent pixels would not be connected, and the receptive fields will always be 1×1. ShiftNets [35] solve this problem using channel shifts, which allows adjacent pixels in different channels to connect through 1×1 convolutions. Channel shift

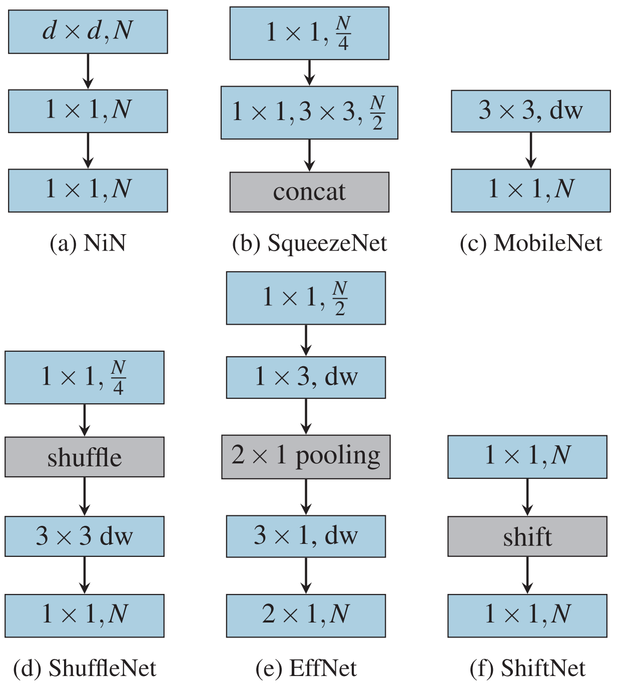

Fig. 3. Sequences of convolutional layers used for fast and compact architectures described in Section 3. The notation follows Fig. 1. In case of ShuffleNet, 1×1 convolutions are also group convolutions

is a cheap operation, but ShiftNet often requires to increase the number of channels in the network to achieve comparable performance. The proposed building blocks of ShiftNet and other architectures listed above are shown in Fig. 3.

The principles of efficient lightweight architecture design are also very useful for the design of large "heavy-weight" architectures. In practical setting, the memory on the GPU and the time for the experiments are always limited, so the depth of the CNN architecture is limited as well. Since the CNN performance usually increases with depth, it is desirable to stretch this limit by designing architectures efficiently. In this arena, the Inception architecture [36] is built on the premise of approximating sparsity with existing building blocks. It was gradually refined [16] to minimize the computational cost and to maximize the performance. Towards this end, convolution decomposition was introduced in the fourth version [37]. Finally, depthwise convolutions were introduced to the Inception architecture in [38]. The ResNet architecture [39] (as well as its futher development ResNeXt [32]) also makes use of similar efficient architecture principles. The training of extremelly deep CNNs (150 layers for ImageNet-sized inputs) simply cannot be done without careful management of computation resources.

Finally, we note once again, that efficient architecture design is not limited to classification tasks, as segmentation and detection require specialized architectures that nevertheless may share the same principles. This subject is outside of scope of this review, but good benchmarks that track both speed and accuracy are provided for the Cityscapes dataset [40] in the case of segmentation and the KITTI benchmark [41] in the case of 2D and 3D detection.

4. Automatic architecture search

Usually, the neural network architectures are hand-constructed by human experts, who are guided by general principles of efficient architecture design. A lot of choices in the architecture construction are left to intuition and guessing. The situation asks for automatic algorithms for architecture search that are reviewed in this section. This section thus naturally complements and expands Section 3. As automated architecture search is a rapidly developing field at the spearhead of modern deep learning research, this section only covers the most influential works in this sub-field.

Automatic architecture search is essentially a hyperparameter optimization, which is a general problem that can be tackled by several approaches, such as grid search or Bayesian optimization [42]. The case of CNN brings several complications. First, the function evaluation becomes very expensive. Second, the number of hyperparameters is very large and may vary across the optimization space. Thus, hyperparameter optimization for CNN requires specialized approaches.

Perhaps the first successful attempt at automatic architecture search is a neural architecture search (NAS) algorithm [43] which utilizes reinforcement learning. The CNN architecture is predicted by a recurrent neural network that sequentially produces CNN hyperparameters: filer sizes, strides, numbers of filters. Possible branchings and skip connections are modelled using an attention mechanism that decides which connection between the current layers and the previous layers are to be introduced in the next step.

NAS requires extremely large amount of computational resources even when performed on small datasets. Thus in the experiments with CIFAR-10 dataset, the authors report testing 12 800 architectures during the search process and using 800 GPUs simultaneously. This makes architecture search for larger datasets such as ImageNet impossible.

Larger datasets can be approached in two ways: by designing more efficient search algorithms, or by ensuring that the results obtained on small dataset are transferrable to the larger datasets. Both approaches are implemented in [44]. First, the modular structure is imposed on a target CNN. This way, only the architecture of a block has to be predicted instead of the whole network, making the search space much smaller. Second, CIFAR-10 and ImageNet versions of architectures can be made of the same blocks with different number of poolings (or strided convolutions) between them. Architectures built this way beat human-constructed architectures on ImageNet.

The neural architecture search can be further accelerated by sharing parameters between different architectures [45, 46] or by predicting the final performance of architecture based on the first epochs in the beginning of the training process [47]. With these enhancement, hundreds GPUs are no longer necessary for automatic architecture search. Thus, [46] reports closely reproducing the original neural archtecture search results using five GPUs instead of 800.

Still, all the variations of approaches which divide architecture construction and evaluation require evaluating thousands of CNNs. The resource consumption can be drastically reduced if the architecture search and CNN training are done at the same time. MorhNets [48] simultaneously learns CNN weights and changes its architecture by iterating between the two stages. In the first stage, the network is thinned by sparsity induced regularizes. In the second stage, new channels are added uniformly to all layers. This algorithm allows CNN to adapt thickness of its layers to the particular task. The running time is comparable with regular CNN training, so it can be directly applied on ImageNet. The downside of this approach is that the search space is limited to changing layer widths.

5. Quantization

A switch to low-precision arithmetic or quantization is a straightforward way to speed up computations, as well as to compress and to minimize memory requirement for a neural network. This general idea produces a spectrum of approaches, which starts from using slightly lower precision and ends with the complete switch to binary weights and activation. While the binarization offers a compelling perspective of extremely low time and memory cost, it is not possible yet to fully transition to binary CNNs without substantial accuracy drops.

The problems faced by quantization and binarization are apparent. First, in the case of quantized weights, it is hard to implement gradient descent, since the idea of quantization contradicts the process of accumulating small changes. Second (and related), if the activations are quantized, the backpropagation process becomes complicated. This section surveys some of the ways to deal with these problems.

The list of approaches for improving the speed of neural networks on CPUs [49] includes 8-bit quantization. Several facts which facilitate quantization are listed. Firstly, because of the sigmoid activation function, activations stay in the $[0, 1]$ interval, so no scaling is needed. Secondly, because of the linear nature of the operation together with range compression by sigmoid, quantization errors tend to propagate sublinearly. That said, modern CNNs rarely use sigmoid activations, so this argument may no longer be valid, although sublinearity of error propagation still holds with ReLU activation. Moreover, [50] notes that neural networks are not only robust towards noise, but the training performance can be enhanced by noise injection. Thus, noise-like distortions injected by the quantization process may not be detrimental if this process is properly tuned. Therefore, if the hardware allows it, low precision arithmetic is a viable technique for speeding up neural networks.

The problem of the compression of fully-connected weights matrix W was addressed in [51]. The compression often goes hand-in-hand with the improvement of the speed, and some techniques are applicable both to convolutional and fully connected layers, so this work turns out to be very influential for speeding up CNNs. The following approaches are compared in [51]:

- SVD-decomposition that approximates weights matrix $W \in \mathbb{R}^{n \times c}$ as a product smaller matrices with truncated SVD:

$$\hat{W} = USV^T = U'V^T \tag{2}$$

where $U, U' \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{c \times k}$. The input multiplication by W is then replaced by two multiplications with U' and V^T. The compression and speed-up rate are controlled by the number of components in the decomposition k.

- Binarization, which is the simplest and the most radical way to compress parameters by applying thresholding:

$$\begin{cases} 1 & W_{ij} > 0 \\ -1 & W_{ij} < 0 \end{cases} \tag{3}$$

Binarization compresses data from full 32 bit precision to 1 bit, with the fixed compression rate of 32×.

- Scalar quantization: here all entries of matrix W are clustered by the k-means algorithm, and the centroid values c_t are used for the approximation

$$\hat{W}_{ij} = c_t \quad \text{where} \quad t = \operatorname*{argmin}_z \left|W_{ij} - c_z\right| \tag{4}$$

- Product quantization [52] divides matrix W into several submatrices $[W_1, W_2 \ldots W_s]$, then each submatrix is clustered by kmeans and compressed independently.

As opposed to the previous results for the convolutional layers, matrix decomposition performed poorly. Simple scalar quantization and product quantization achieved much better results, and, surprisingly, the simplest binarization techinque also worked reasonably well. Simplicity and high compression rate make binarization a very promising approach, but its capabilities are limited unless one can properly train binarized networks and recoup the accuracy drop incurred by binarization. A multitude of attempts at reconciling binarization with backpropagation and gradient descent were done in the recent years.

The initial development of algorithms for neural network quantization was mostly done in the area of speech recognition, which is outside of the scope of this work. For images, [53] presents an algorithm for training quantized CNNs. The basic idea is to keep two versions of the weights: quantized $\hat{W}$ and full precision W. The algorithms repeats the following steps:

1. Obtain quantized weights with some sort of quantization procedure q applied to high-precision weights:

$$\hat{W} = q(W) \tag{5}$$

2. Perform a feed-forward pass with quantized weights and compute the loss function. The activations are kept in full precision.
3. Backpropagate the error gradients with quantized weights and full precision activations. The gradients are then used to update the full-precision weights.

This sequence allows to circumvent the problems with backpropagation and gradient descent for quantized weights. Different quantization levels are used for different layers and the main benefit here is the model compression.

The presented results on MNIST and CIFAR10 datasets exceed uncompressed CNNs in some cases. This effect is attributed to the regularizing effect of quantization which reduces the CNN capacity (similar effect was reported in [50]). At the same time, there are other well-studied ways to reduce capacity, such as regularization, dropout, and simply reducing the number of filters inside CNN. The latter method also leads to speed-up and compression. Overall, the main disadvantage of [53], shared with many others in the field, is that the experiments are limited to small networks with 32×32 inputs. Such CNNs have relatively low capacity and are quick to experiment with. And yet, the main challenge of speeding up CNNs lies in the area of big CNNs (such as those trained for ImageNet classification and other similar tasks), to which good results on small images do not always transfer.

The BinaryConnect [54] approach pushes the principle of splitting high-precision and quantized weights further to achieve full binarization, i.e. training networks with binary weights in the convolutionals tensors. The binarization procedure (3) is modified in a probabilistic fashion:

$$\begin{cases} 1 & \text{with probability} & p = \sigma(w) \\ -1 & \text{with probability} & 1 - p \end{cases} \tag{6}$$

where σ is the hard version of sigmoid function

$$\sigma(x) = \operatorname{clip}\left(\frac{x+1}{2}, 0, 1\right) \tag{7}$$

which is chosen because it is much less computationally expensive compared to the regular sigmoid function. The high-precision weight are clipped into $[-1; +1]$ interval during training. A probabilistic approach is in general desirable from theoretical point of view. On the other hand, the cost of random number generation accumulates if used on every step.

At train time, BinaryConnect repeats the following steps:

1. For every high-precision weight w, pick $+1$ or -1 according to (6).
2. Do feed-forward pass with binarized weights $\hat{w}$.
3. Do backpropagation with binarized weights $\hat{w}$ and update high-precision weights w.

At test time, naturally, only the binarized weight are used since only the forward propagation is needed. The paper [54] proposes generating binarized weights multiple times as test time to obtain the ensemble of models, but ensembling contradicts with the speed-up task and preserving full-precision weights contradicts compression. Forward propagation with binary weights can be much faster since it replaces floating-point multiplications by multiplications with ± 1, which is just a sign change. Competitive results are presented for CIFAR10, SVHN and permutation-invariant MNIST. Again, as in the case with [53], classification accuracy sometimes exceeds the full-precision baseline for small datasets.

Two extensions of this approach are presented in the follow-up paper [55]. First of all, ternary weight are introduced. Ternary weights are obtained by a stochastic procedure similar to (6). Every weight w is assumed to lie in the interval $[-1; 1]$. This interval is divided into two sub-intervals $[-1; 0]$ and $[0; 1]$ and the probability of picking 1, 0 or -1 is determined by the procedure (6) applied to the respective interval. The second important innovation is the elimination of multiplications in backward pass. The layer activations are quantized into 3 or 4 bits and multiplication is replaced by bit-shifts.

The comparison of learning curves shows that binary and ternary networks behave similarly both with and without backward pass quantization: initially, convergence is slower, but the final result can be better. Again, this is attributed to the regularizing effect of quantization.

Subsequent paper on binarized neural networks [56] provides the details on the practical implementation and timings. Shift-based versions for batch normalization [57] and ADAM optimization [58] algorithm are presented. A custom CUDA kernel is written for binary matrix multiplication, and its speed is compared to cuBlas on 8192×8192 matrix multiplication. The binary kernel is reported to be $3.4\times$ faster. The preliminary results on the ILSVRC challenge with AlexNet architecture is 36.1% top-1 accuracy, which corresponds to $\sim20\%$ accuracy drop compared to the full-precision architecture.

Another interesting and more successful attempt at binarizing large CNNs is the XNOR-Net [59]. Here, the weight binarization setting is considered as an approximation problem of the following kind:

$$I*W \simeq \alpha(I*B), \tag{8}$$

where $*$ denotes convolution, I is the input tensor, α is a scaling factor, W is high-precision weights tensor and B is its binary version. It can be shown that the optimal values of elements of W are indeed obtained by the simple binarization procedure (3), and the optimal value for the scaling factor is an average of the absolute values of W. Training the binary weights network is done by repeating the same tree steps from [53, 54], i.e. obtaining binarized weights from high precision weights, doing forward and backward passes with binarized weights, and applying updates to the full precision weights.

The next step is to binarize both weights and activation, resulting in the so-called *XNOR-networks*. The approximation problem of the following form is considered:

$$I*W \simeq \big(\operatorname{sign}(I)*\operatorname{sign}(W)\big)\odot K\alpha, \tag{9}$$

where $\odot$ is an element-wise multiplication and K is a tensor with scaling factors for every patch in I. This approximation leads to $\times58$ speed-up in terms of the number of the floating point operations, while the actual timings will depend on the cost of binary operations, which depends on the hardware and implementation details.

XNOR-Nets training also requires the following rearrangement of the traditional CNN block sequence, in order to minimize the information loss in binarization:

1. Batch normalizations are put in the beginning of the block.
2. Following batch normaliztion, the binary activation layer computes K and $\operatorname{sign}(I)$.
3. Binary convolution is applied to the result of the binary activation layer.
4. Optionally, pooling is applied.

The blocks of layers composed in the same way are then applied several times.

Overall, the accuracy of binarized XNOR-networks is shown in the Table 2. Interestingly, the accuracy for full-precision AlexNet and its version with binarized weights is the same, although this effect does not hold for larger architectures. Thus for ResNet-18, XNOR-Net loses more then 10% of accuracy compared to the full-precision network.

Table 2
ImageNet (ILSVRC) classification accuracy of binarized CNNs from [59]. The accuracy drop is large compared to tensor decomposition methods, but the speed-up and compression rates assosiated with binarization are much higher

	full precision	binary weights	XNOR-Nets
AlexNet	56.6	56.8	44.2
ResNet-18	69.3	60.8	51.2

Generally speaking, binarization is an approach with low flexibility: it promises extremely large speedups, but often incurs substantial accuracy drop which may be unacceptable in practical applications. One way to cover this gap is to dial compression rate back and return from binarization to low-bit quantization. Towards this end, quantization with different compression rates is considered in [60]. Quantized version of AlexNet with 1-bit weights and 2-bit activations achieves 51% accuracy. Varying quantization levels for weights, activations and gradients are tried in [61]. Another way to boost accuracy of binarized CNN is by increasing number of channels. This approach was found beneficial in [62].

Yet another quantization based approach with more flexibility is Lookup-based CNN (LCNN) introduced in [63]. Interestingly, LCNN utilizes ideas of decomposition, quantization and sparsity at the same time. First of all, the convolutional kernel W is decomposed into the sum of vectors of the dictionary matrix D, with coefficients C and indices I:

$$W(i,j,:,t) = \sum_{\xi=1}^{s} C(\xi,i,j)D\big(I(\xi,i,j),:\big) \tag{10}$$

The two spatial dimensions and the last dimension corresponding to the output channel index are not affected by the decomposition. Following this insight, one may take advantage of the fact that the general convolution can be expressed trough 1×1 convolutions and shift operations:

$$V(x,y,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{s=1}^{S} W(i,j,s,t)\Big[\underset{ij}{\operatorname{shift}}U\Big](x,y,s), \tag{11}$$

where $\operatorname{shift}_{i,j}$ indicates the spatial shift operation. Combining (10) and (11) yields the following way to perform generalized convolutions:

$$V(x,y,t) = \sum_{i=0,j=0}^{d,d} \operatorname{shift}_{i,j}\sum_{\xi=1}^{s} C(\xi,i,j)S \tag{12}$$

where the tensor S contains the result of 1×1 convolutions of the input U with the filters from D. Shifts, scaling and 1×1 convolution, which is implemented through matrix multiplication, are all relatively inexpensive multiplication. The cost of this pipeline can be regulated by changing the dictionary size.

Direct training of the proposed lookup based convolution is a combinatorial optimization problem. To get around this, the lookup and scale stage are reformulated using a standard convolution with sparsity constraints. Reported speedups of LCNN reach $37.6\times$, as shown in the Table 3. However, this value refers to the number of floating point operations, which may not translate well to actual timings, especially for the architecture that heavily relies on the lookups (which are known to be relatively slow on most architectures).

Table 3
LCNN accuracy on ImageNet (ILSVRC) classification task. The performance of LCNN can be tuned by changing dictionary size and the number of components in the decomposition. Two variants of the algorithm are shown in this table. The speed-ups are measured in terms of FLOPs, and the actual "wall-clock" speed-ups are likely to be much lower on most architectures

	AlexNet		ResNet-18	
	accuracy	speedup	accuracy	speedup
CNN	56.6	$1\times$	69.3	$1\times$
LCNN-accurate	55.4	$3.2\times$	62.2	$5\times$
LCNN-fast	44.3	$37.6\times$	51.8	$29.2\times$

To summarize, weight quantization or binarization is an effective technique for CNN compression. As for the speed-up, the published works paint a mixed picture. It is clear that quantization of CNN weights or activations allows for faster computations, but the actual speed up depends on a particular low-level implementation. Most of the time researchers do not publish such implementations, and when they do, it appears that existing implementations of floating point operations are very well optimized and the actual speed ups brought by quantization methods are not nearly as high as the operation-count based prediction suggests.

6. Pruning

Pruning away parts of the convolutional tensor is a natural way to reduce the complexity of the convolutional operation. This approach, applied for speeding up convolutions in neural networks, is a popular research topic with a very large number of publications. Starting from the optimal brain damage [64], this is perhaps the oldest approach among listed in this review.

Most of the pruning approaches follow the same pipeline. Starting from the pretrained baseline, the following two steps are applied, possibly iteratively. First, the importance of neurons is calculated according to some criterion. The least important neurons are pruned. Then, the network is fine-tuned leading to partial recovery of the accuracy drop. In the case of the iterative process, sparsity inducing regularizer may be applied during the fine-tuning stage.

Three basic choices have to be made to implement this pipeline. First, the desired sparsity structure must be chosen. Second, the importance (pruning) criterion should be selected. Finally, a sparsity-inducing regularizer should be chosen (if the approach uses one). Below, we review different design choices along these three axes.

6.1. Sparsity structure. We first note that pruning individual weights does not necessarily results in a speed-up. Assume the convolution is implemented through `im2col` and matrix multiplication, as described by [65]. In this implementation, most of computation time is spent inside the matrix multiplication

$$\hat{V} = \hat{W}\hat{U} \tag{13}$$

where $\hat{V}$ is the output in the matrix-reshaped form, $\hat{W}$ is the convolutional weights tensor also reshaped as a matrix and $\hat{U} = \mathrm{im2col}(U)$ is the patch matrix obtaining by copying and rearranging of input tensor U by the im2col operation. The columns of the patch matrix correspond to input tensor patches of size $d_x \times d_y \times S$.

As some elements of W will be replaced by zeros by the pruning algorithm, one can switch to some sparse representation for W. However, in the lack of the structure of the sparsity, sparse matrix multiplication carries significant overhead compared to dense matrix multiplication. Usually, sparse version becomes faster only if the density of W is below 0.1, which is unreachable in practical setting without a significant drop in accuracy. The only way to overcome this problem is to arrange elements of W into groups and to use structured sparsity.

The consideration discussed above calls for the use of structured sparsity during pruning. The finest possible division into groups is considered in [66] and [67]. Their algorithms remove columns from weight matrix $\hat{W}$ and corresponding rows from patch matrix $\hat{U}$. The removal is facilitated by the custom version of the `im2col` function which omits elements corresponding to deleted parts of $\hat{W}$ while constructing the patch matrix. With this kind of structured sparsity, the original matrix multiplication is replaced with multiplication of smaller matrices, which are still dense. This leads to the speed ups that are almost directly proportional to density.

A related but orthogonal approach to structured sparsity called *perforation* was proposed in [68]. The main idea is to remove columns from the patch matrix $\hat{U}$. Since columns correspond to image patches, this means the convolution will not be computed for some subsets of points in the image. The output value in these points can be interpolated or effectively omitted if the next layer performs the pooling operation.

Several additional ways to organize sparsity are proposed in [69] and [70]. Each of them is defined by the specific way of slicing the four dimensional weight tensor W:

- Slicing in the form of $W(i, j, s, :)$ is the same as in the groupwise sparsity approach and produces non-square filters. It is the finest division that can be implemented efficiently, but it requires specialized implementation.

- Removing $W(i, j, :, :)$ cuts all filters in the layer simultaneously. This slicing can be used to trim the filter size, for example from 5×5 to 3×3.
- A whole filter corresponds to slice $W(:, :, s, :)$. With such slice set to zero, the s-th channel in the output tensor will be filled with zeros. The complexity of the network then can be decreased by removing slices from weight tensors of the current and the next convolutional layers.
- Removing $W(:, :, :, t)$ cuts all the connection with t-th input channel, which means this channel can be removed.
- Removing $W(:, :, s, t)$ cuts all the ties between the s-th input and the t-th output channels. This slicing can be used to turn full convolution into group convolution [7].
- Finally, in case of residual architecture, the convolutional tensor W can be set to zeros completely. This operation removes the whole residual block of the network.

5.2. Pruning criteria. Here, we follow [71], which contains a similar review of criteria, and denote the pruning criterion by Θ. The simplest criterion is the absolute value of the weight:

$$\Theta(w) = |w|. \tag{14}$$

It was succesfully used in [72] for pruning individual weights, and then in [18] for groups. This criterion is consistent in the sense that if the weight already equals zero it can be safely pruned, but small nonezero weight can be disproportionally important if it acts on a large activation or pushes some points across the decision surface.

Another choice is to focus not on the weights, but on the **activations** a:

$$\Theta(a) = \sum_i a_i^2. \tag{15}$$

Since ReLU non-linearity naturally produces sparse activations, there is a realistic chance to find groups of neurons which can be safely pruned. Average percentage of zeros is a different metric proposed in [73] for this situation.

Mutual information measures the dependence of two random variables. In theory, mutual information between activation group and targets $I(a, y)$ would be an excellent pruning criterion, but the direct computation is too complex, and available approximations are not performing well according to [71].

Taylor expansion of the objective function can be used to estimate its change after the perturbation caused by the pruning process. For example, the original Optimal Brain Damage paper [64] used the criterion based on the second-order Taylor decomposition. Assuming that C is the learning objective, the approach makes an assumption that $\frac{\partial C}{\partial w_i} = 0$ ("extremal approximation"), which holds when the learning has fully converged. Assuming that the mixed derivatives could be neglected, the approach then uses the non-mixed second-order derivatives as the pruning criterion:

$$\Theta(w_i) = \frac{1}{2}\frac{\partial^2 C}{\partial w_i^2} w_i. \tag{16}$$

The necessity to compute second derivatives makes this method inconvenient in practice. The works [68, 71] avoid the extremal approximation and use the following criterion:

$$\Theta(a_i) = \left|\frac{\partial C}{\partial a_i} a_i\right|. \tag{17}$$

This criterion is expressed through the values which can be computed by the standard back-propagation process. In general, a comparison of pruning criteria listed above performed by [71] demonstrated a superiority of the Taylor-expansion based criteria.

The ThiNet approach [74] focuses on pruning filters and proposes a special criterion for this case. The key observation is that if the filter is pruned from the i-th layer, the corresponding output channel will be empty, and the same channel should be pruned in the kernel of the $(i+1)$-th layer. The next, $(i+2)$-th layer will then be the first subsequent layer, whose input tensor size is not affected by the change. Thus, a natural pruning criterion relies on the reconstruction error of the input tensor to the $(i+2)$-th layer. After the pruning, the channel scaling computed via least squares can be used to reduce the error, although this step cannot replace the fine-tuning.

6.1. Regularizer. The pruning process can work without sparsity-inducing regularization, but the sparsity-inducing regularization can help the pruning process while incurring a minimal computational overhead. The $L1$ reguarizer $\Omega_1(w) = \lambda|w|$ induces unstructured sparsity, but for structured sparsity, $L2, 1$ regularization can be used:

$$\Omega_{2,1}(w) = \lambda \sum_i \sqrt{\sum_{j \in gi} w_i}\,. \tag{18}$$

Here, g_i are the weight groups, defined by one of the ways described above. A smart approach for achieving filter-level sparsity was proposed in [75]. They notice that in modern CNNs, convolutions are almost always followed by batch normalization. The filter level sparsity can then be achieved simply by the L_1 regularization imposed on the scaling factors within batch normalization.

7. Teacher-student approaches

Teacher-student approaches follow the idea that a CNN model can be trained on the outputs of another model (a teacher), as opposed to regular training on labeled data. This approach allows to transfer knowledge from one model to another and to incorporate unlabelled or synthetic data into the training process (as an unlabeled example can still be passed through the pretrained teacher model). Originally, such transfer was performed from a non-interpreTable model such as a neural network, to more interpreTable ones, such as decision trees [76], or a set of rules [77]. Another natural purpose for the teacher-student approach would be to transfer knowledge from large, slow and accurate models to small and fast ones.

Towards this end, [78] proposes to compress an ensemble of models into a single neural network. First, an ensemble of classification models is trained on a certain annotated dataset. An ensemble is expected to be more resistant to overfitting compared to a single model. This ensemble is used to label a large amount of synthetic data, generated by several simple random sampling procedures, and finally a single model is learned on the resulting synthetic dataset. Authors of [78] state that this approach can alleviate overfitting problem for neural networks without time and memory costs of building an ensemble. It should be noted, though, that this work was done on small datasets with fully-connected neural nets, and utilized data generation methods that are not directly applicable to images. With modern CNNs, the viability of this approach is limited by the following facts: datasets are already very large and the models are too large to build large ensembles.

A specific way of representing labels for synthetic data is a key detail of knowledge transfer algorithm. The description of same ensemble compression idea in [79] elaborates on the importance of preserving not just the labels, but whole vectors of posterior probability distribution over the output classes. Such vectors capture richer information about the actual content of data samples, making knowledge transfer process more efficient.

This idea of utilizing full probability distribution is further expanded in [80]. The proposed distillation procedure requires training a student model on the soft version of the outputs of the original (teacher) model. Let z_i be the raw outputs of neural network, and p_i be the output probabilities. These probabilities are then calculated according to the softmax formula:

$$p_i = \frac{\exp z_i/\tau}{\sum_j \exp z_j/\tau} z, \tag{19}$$

where τ is the temperature parameter. Let p^S be the probabilities for the student model, and p^T be the probabilities of original teacher model. The student model is trained to approximate both the correct labels y and the outputs of teacher model using the following loss function:

$$L = H\left(p^S, p^y\right) + \lambda H\left(p^S, q^T\right), \tag{20}$$

where H is the the cross-entropy and p^y is the one-hot distribution corresponding to the ground truth. A high temperature τ effectively regularizes the student model, while the lower temperature allows to transfer knowledge in finer detail. In practice, the temperature parameter τ has to be tuned manually. It can be shown that in the limit of high temperature this procedure is equivalent to training on raw outputs z_i (the regime which was utilized in [81] for acoustic model compression). Results on MNIST, an automatic speech recognition task and large scale image classification task are presented, and significant rate of model compression is achieved for all tasks. Most impressively, it is shown that knowledge transfer can be successful even if one of the classes is missing from the dataset used for the transfer, since the information about this class is still carried through soft labels of other classes.

The idea of distillation was extended to multiple layers of deep networks in the Fitnets approach [82], which introduces the notion of a hint, defined as the output u_T of the teacher's hidden layer, and used to guide the training of a hidden layer of the student CNN, called the guided layer. The guidance process is implemented via addition of the euclidean distance between the hint and the guided layer output u_S to the loss function (20). When the layer sizes of the hint and the guided layer differ, the linear regressor r that maps the hints to the guided layer activations is added into the training leading to the following term (the guidance loss):

$$L_h = \frac{1}{2}\left\| u_T - r\left(u_S\right)\right\|^2. \tag{21}$$

The fact that the student now not only has access to the outputs of the teacher, but also receives insights from the internal data representation, leads to faster convergence and better performance of the method.

Teacher-student approach is a powerful tool which can be used to help training of quantized networks. It has been successfully applied to training of quantized networks [83] and ternary networks [84]. Moreover, while most papers focus on the situation where both the teacher and the student are neural networks, the need for the faster student may lead to different kinds of models. Thus, [85] proposes to use soft decision tree as a student model. Decision tree in theory can provide very high speed-ups, but in this paper only results on MNIST and Connect4 datasets are presented, and the achieved MNIST accuracy of 96.76% is below modern standards.

8. Discussion

As can be seen from this non-exhaustive survey, a wide variety of approaches has been tried to speed up CNNs. The modern CNN architectures are more efficient than ones available a few years ago, but still a lot of improvement is required to use them fully on low-end hardware and/or real-time constraints.

Some approaches, such as tensor decomposition based methods, seem to reach saturation, as virtually all decompositions have been tried. With other approaches, e.g. those based on binarization, there is a lot of work on algorithms and implementation that can be done in future. Here, transferring impressive speed-ups in terms of number of operations into actual wall clock speed-ups remains a challenge.

Designing efficient architectures is probably the most practical approach, as it does not require complex multi-stage processes that interleave modifications and finetuning stages. While it seems that in the future these architectures will be constructed automatically, some basic modules or design ideas are likely to come from humans rather than from automated search. Still, while automatic architecture search is a young area of research, it has already made an impact and it is safe to assume that it will continue to grow in importance in the nearest future.

There is probably a reasonable space for the search of optimal combination of approaches from several groups, notably from those that speed-up existing architectures (tensor decomposition, quantization, pruning, teacher-student approaches). While these groups are not "orthogonal" as they exploit similar kind of redundancy in the original architecture, there may still be considerable benefits in combining approaches from different groups. Automated discovery of optimal mix-and-match combinations may be promising.

Overall, we believe that in general the area of research concerned with speeding up CNNs (as well as designing efficient architectures "from scratch") is far from saturation, and significant improvements can and will be made in the nearest future.

Acknowledgements. This work is supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001)

References

[1] T. Young, D. Hazarika, S. Poria, and E. Cambria, Recent trends in deep learning based natural language processing, *arXiv preprint arXiv:1708.02709*, 2017.

[2] S. Min, B. Lee, and S. Yoon, Deep learning in bioinformatics. *Briefings in bioinformatics*, 2017.

[3] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, Benchmarking deep reinforcement learning for continuous control, In *ICML*, 2016.

[4] K. Fukushima and S. Miyake, Neocognitron: A selforganizing neural network model for a mechanism of visual pattern recognition, *Competition and cooperation in neural nets*, 1982.

[5] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural computation*, 1989.

[6] R. Raina, A. Madhavan, and A.Y. Ng, Largescale deep unsupervised learning using graphics processors, In *ICML*, 2009.

[7] A. Krizhevsky, I. Sutskever, and G.E. Hinton, Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[8] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu, Recent advances in efficient computation of deep convolutional neural networks. *arXiv preprint arXiv:1802.00939*, 2018.

[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J.S. Emer, Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 2017.

[10] R.K. Srivastava, K. Greff, and J. Schmidhuber, Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[11] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K.Q. Weinberger, Deep networks with stochastic depth. In *ECCV*, 2016.

[12] M. Figurnov, M.D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, Spatially adaptive computation time for residual networks. *arXiv preprint*, 2017.

[13] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, Adaptive neural networks for efficient inference. In *ICML*, 2017.

[14] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, Learning separable filters. In *CVPR*, 2013.

[15] M. Jaderberg, A. Vedaldi, and A. Zisserman, Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[17] E. Denton, W. Zaremba, J. Bruna, Y. Le-Cun, and R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation. *arXiv preprint arXiv:1404.0736*, 2014.

[18] V. Lebedev, Y. Ganin, M. Rakhuba, I.V. Oseledets, and V.S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *ICLR*, 2015.

[19] M. Astrid and S.-I. Lee, Cp-decomposition with tensor power method for convolutional neural networks compression. In *Big Data and Smart Computing*, 2017.

[20] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM Rev*., 2009.

[21] G. Allen, Sparse higher-order principal components analysis. In *AISTATS*, 2012.

[22] J. Jin, A. Dundar, and E. Culurciello, Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[23] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *TPAMI*, 2016.

[24] 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. *arXiv preprint arXiv:1511.06530*, 2015.

[25] P. Wang and J. Cheng, Accelerating convolutional neural networks for mobile applications. In *ACM Multimedia*, 2016.

[26] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, Tensorizing neural networks. In *NIPS*, 2015.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in pytorch. *ICLR*, 2017.

[29] M. Lin, Q. Chen, and S. Yan, Network in network. *ICLR*, 2014.

[30] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, and K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[31] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[32] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[34] I. Freeman, L. Roese-Koerner, and A. Kummert, EffNet: An Efficient Structure for Convolutional Neural Networks. *arXiv preprint arXiv:1801.06434*, 2018.

[35] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, Shift: A zero flop, zero parameter alternative to spatial convolutions. *arXiv preprint arXiv:1711.08141*, 2017.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al., Going deeper with convolutions. In *CVPR*, 2015.

[37] C. Szegedy, S. Ioffe, and V. Vanhoucke, Inception-v4, inception-resnet and the impact of residual connections. *AAAI*, 2017.

[38] F. Chollet, Xception: Deep learning with depthwise separable convolutions. *CVPR*, 2017.
[39] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition. *CVPR*, 2016.
[40] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.
[41] A. Geiger, P. Lenz, and R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.
[42] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, and N. De Freitas, Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 2016.
[43] B. Zoph and Q.V. Le, Neural architecture search with reinforcement learning. *ICLR*, 2017.
[44] B. Zoph, V. Vasudevan, J. Shlens, and Q.V. Le, Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.
[45] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, and J. Dean, Efficient neural architecture search via parameter sharing. *ArXiv*, 2018.
[46] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, Reinforcement learning for architecture search by network transformation. *ArXiv preprint ArXiv:1707.04873*, 2017.
[47] B. Baker, O. Gupta, R. Raskar, and N. Naik, Practical neural network performance prediction for early stopping. *ArXiv preprint ArXiv*:1705.10823, 2017.
[48] A. Gordon, E. Eban, O. Nachum, B. Chen, T.-J. Yang, and E. Choi, Morphnet: Fast & simple resourceconstrained structure learning of deep networks. *ArXiv preprint ArXiv:1711.06798*, 2017.
[49] V. Vanhoucke, A. Senior, and M.Z. Mao, Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.
[50] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv preprint ArXiv:1207.0580*, 2012.
[51] Y. Gong, L. Liu, M. Yang, and L. Bourdev, Compressing deep convolutional networks using vector quantization. *ArXiv preprint ArXiv:1412.6115*, 2014.
[52] H. Jegou, M. Douze, and C. Schmid, Product quantization for nearest neighbor search. *TPAMI*, 2011.
[53] S. Anwar, K. Hwang, and W. Sung, Fixed point optimization of deep convolutional neural networks for object recognition. In *ICASSP*, 2015.
[54] M. Courbariaux, Y. Bengio, and J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations. *NIPS*, 2015.
[55] Z. Lin, M. Courbariaux, R. Memisevic, and Y.a Bengio, Neural networks with few multiplications. *ICLR*, 2016.
[56] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, Binarized neural networks. *NIPS*, 2016.
[57] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv preprint ArXiv:1502.03167*, 2015.
[58] D.P. Kingma and J. Ba, Adam: A method for stochastic optimization. *ICLR*, 2014.
[59] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.
[60] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations. *ArXiv preprint ArXiv:1609.07061*, 2016.
[61] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *ArXiv preprint ArXiv:1606.06160*, 2016.
[62] A.K. Mishra, E. Nurvitadhi, J.J. Cook, and D. Marr, WRPN: wide reduced-precision networks. *ArXiv preprint ArXiv:1709.01134*, 2017.
[63] H. Bagherinezhad, M. Rastegari, and A. Farhadi, Lcnn: Look-up-based convolutional neural network. *CVPR*, 2017.
[64] Y. LeCun, J.S. Denker, and S.A. Solla, Optimal brain damage. In *NIPS*, 1990.
[65] K. Chellapilla, S. Puri, and P. Simard, High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.
[66] V. Lebedev and V. Lempitsky, Fast convnets using group-wise brain damage. In *CVPR*, 2016.
[67] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, Sparse convolutional neural networks. In *CVPR*, 2015.
[68] M. Figurnov, A. Ibraimova, D.P Vetrov, and P. Kohli, Perforatedcnns: Acceleration through elimination of redundant convolutions. In *NIPS*, 2016.
[69] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, Learning structured sparsity in deep neural networks. In *NIPS*, 2016.
[70] R. Shin, C. Packer, and D. Song, Differentiable neural network architecture search, 2018.
[71] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, Pruning convolutional neural networks for resource efficient transfer learning. *ArXiv preprint ArXiv:1611.06440*, 2016.
[72] S. Han, J. Pool, J. Tran, and W. Dally, Learning both weights and connections for efficient neural network. In *NIPS*, 2015.
[73] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *ArXiv preprint ArXiv:1607.03250*, 2016.
[74] J.-H. Luo, J. Wu, and W. Lin, Thinet: A filter level pruning method for deep neural network compression. *CVPR*, 2017.
[75] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, Hierarchical representations for efficient architecture search. *ArXiv preprint ArXiv:1711.00436*, 2017.
[76] M. Craven and J.W. Shavlik, Extracting tree-structured representations of trained networks. In *NIPS*, 1996.
[77] S. Thrun, Extracting rules from artificial neural networks with distributed representations. In *Advances in neural information processing systems*, 1995.
[78] C. Bucila, R. Caruana, and A. Niculescu-Mizil, Model compression. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.
[79] X. Zeng and T.R. Martinez, Using a neural network to approximate an ensemble of classifiers. *Neural Processing Letters*, 2000.
[80] G.E. Hinton, O. Vinyals, and J. Dea, Distilling the knowledge in a neural network. *NIPS 2014 Deep Learning Workshop*, 2014.
[81] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, Learning small-size dnn with output-distribution-based criteria. In *Interspeech*, 2014.
[82] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, Fitnets: Hints for thin deep nets. *ICLR*, 2015.
[83] A. Polino, R. Pascanu, and D. Alistarh, Model compression via distillation and quantization. *ICLR*, 2018.
[84] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, Ternary neural networks for resource-efficient ai applications. In *IJCNN*, 2017.
[85] N. Frosst and G. Hinton, Distilling a neural network into a soft decision tree. *ArXiv preprint ArXiv:1711.09784*, 2017.