# Towards Evolutionary Compression

**Conference Paper** · July 2018

DOI: 10.1145/3219819.3219970

**5 authors**, including:

Yunhe Wang
Huawei Technologies
**65** PUBLICATIONS **219** CITATIONS
SEE PROFILE

Jiayan Qiu
The University of Sydney
**11** PUBLICATIONS **18** CITATIONS
SEE PROFILE

# Towards Evolutionary Compression

Yunhe Wang
Key Lab. of Machine Perception (MOE), Cooperative Medianet Innovation Center, School of EECS, Peking University
Beijing, China
wangyunhe@pku.edu.cn

Chang Xu
UBTECH Sydney AI Centre, SIT, FEIT, University of Sydney
Darlington, Sydney, Australia
c.xu@sydney.edu.au

Jiayan Qiu
UBTECH Sydney AI Centre, SIT, FEIT, University of Sydney
Darlington, Sydney, Australia
jqiu3225@uni.sydney.edu.au

Chao Xu
Key Lab. of Machine Perception (MOE), Cooperative Medianet Innovation Center, School of EECS, Peking University
Beijing, China
xuchao@cis.pku.edu.cn

Dacheng Tao
UBTECH Sydney AI Centre, SIT, FEIT, University of Sydney
Darlington, Sydney, Australia
dacheng.tao@sydney.edu.au

## ABSTRACT

Compressing convolutional neural networks (CNNs) is essential for transferring the success of CNNs to a wide variety of applications to mobile devices. In contrast to directly recognizing subtle weights or filters as redundant in a given CNN, this paper presents an evolutionary method to automatically eliminate redundant convolution filters. We represent each compressed network as a binary *individual* of specific fitness. Then, the *population* is upgraded at each evolutionary iteration using genetic operations. As a result, an extremely compact CNN is generated using the fittest individual, which has the original network structure and can be directly deployed in any off-the-shelf deep learning libraries. In this approach, either large or small convolution filters can be redundant, and filters in the compressed network are more distinct. In addition, since the number of filters in each convolutional layer is reduced, the number of filter channels and the size of feature maps are also decreased, naturally improving both the compression and speed-up ratios. Experiments on benchmark deep CNN models suggest the superiority of the proposed algorithm over the state-of-the-art compression methods, *e.g.* combined with the parameter refining approach, we can reduce the storage requirement and the floating-point multiplications of ResNet-50 by a factor of 14.64× and 5.19×, respectively, without affecting its accuracy.

## CCS CONCEPTS

• **Computing methodologies → Neural networks**; *Supervised learning by classification*; • **Mathematics of computing** → Network optimization;

## KEYWORDS

deep learning; evolutionary algorithm; network compression; CNN acceleration

**ACM Reference Format:**
Yunhe Wang, Chang Xu, Jiayan Qiu, Chao Xu, and Dacheng Tao. 2018. Towards Evolutionary Compression. In *KDD 2018: 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219970

## 1 INTRODUCTION

Large-scale deep convolutional neural networks (CNNs) have been successfully applied to a wide variety of applications such as image classification [15, 24, 30, 36–38], object detection [11, 33], and visual enhancement [9]. To strengthen representation capability and improve CNN performance, several convolutional layers have traditionally been used in network construction. Given the complex network architecture and numerous variables, most CNNs place excessive demands on storage and computational resources, thus limiting them to high-performance servers.

We are now in an era of intelligent mobile devices. Deep learning, one of the most promising artificial intelligence techniques, is expected to reduce reliance on servers and to apply advanced algorithms to smartphones, tablets, and wearable computers. Nevertheless, it remains challenging for mobile devices without GPUs and the necessary memory to carry CNNs usually running on servers. For instance, more than $232MB$ of memory and $7.24 \times 10^8$ floating number multiplications would be consumed by AlexNet [24] or VGGNet [36] to process a single, normal-sized input image. Hence, special developments are required to translate CNNs to smartphones and other portable devices.

To overcome this conflict between reduced hardware configurations and the higher resource demands of CNNs, several attempts have been made to compress and speed up well-trained CNN models by refining their parameters including weight pruning [2, 13, 14, 49], quantization and binarization [1, 5, 6, 20], and matrix decomposition [8]. Wherein, pruning based methods achieved the highest

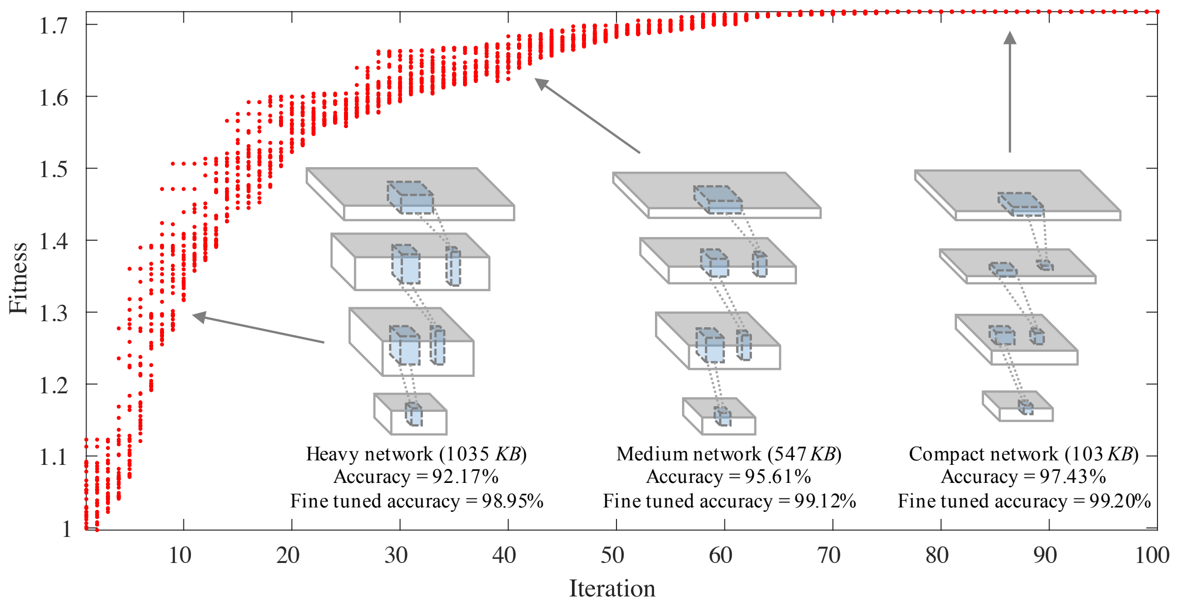


**Figure 1: An illustration of the evolution of LeNet on the MNIST dataset. Each dot represents an *individual* in the population, and the thirty best individuals are shown in each evolutional iteration. The fitness of individuals is gradually improved with an increasing number of iterations, implying that the network is more compact but remaining the same accuracy. The size of the original network is about 1.5*MB*.**

compression performance. In specific, Liu *et al.* [27] developed to learn CNNs with sparse architectures, thereby reducing model complexities compared to ordinary CNNs, while Han *et al.* [14] directly discarded subtle weights in pre-trained CNNs to obtain sparse CNNs. In the DCT frequency domain, Wang *et al.* [44] excavated redundancy on all weights and their underlying connections to deliver higher compression and speed-up ratios. In addition, there also some works to refine the architecture of the pre-trained CNN and construct portable neural networks. For instance, the teacher-student paradigm [17, 34, 45, 50] learns a deeper student network supervised by original CNNs with different regularizations. Wang *et al.* [42] excavated redundancy in feature maps generated by considerable convolution filters and then reconstructed a portable network.

Although the above mentioned methods for eliminating redundancy in convolution filters have reduced the storage and computational burdens of CNNs to some extend, the research on deep model compression is still in its infancy. Existing solutions are typically grounded in different, albeit intuitive, assumptions of network redundancy, *e.g.* weight or filter redundancy with small absolute values, low-rank filter redundancy, and within-weight redundancy. Although these redundancy assumptions are valid, we hypothesize that all possible types of redundancy have yet to be identified and validated. For example, a large weight may be connected with extremely small input data, which has mere effect on the entire network. In addition, compressed CNN models produced by most of parameter refining methods [8, 14, 44] have their own structures and calculation methods, which cannot be directly deployed using conventional toolboxes and hardwares such as Caffe, Tensorflow, and GPU devices. In specific, the convolution operation in these compressed networks are implemented using complex tricks such as sparse kernels, matrix decomposition, and coefficient representation, which leads to their theoretical accelerations are difficult to achieve. By comprehensively and thoroughly investigating the diverse and volatile network redundancies, we expect a compressed model that is stored in regular network network formats and can be constantly upgraded to cater for environment changes.

In this paper, we develop an evolutionary strategy to excavate and eliminate redundancy in deep neural networks as illustrated in Fig. 1. The network compression task can be formulated as a binary programming problem, where a binary variable is attached to each convolution filter to indicate whether or not the filter takes effect. Inspired by studies in evolutionary computation [7, 23, 32], we treat each binary encoding w.r.t. a compressed network as an individual and stack them to constitute the population. A series of evolutionary operators (*e.g.* crossover and mutation) allow the population to constantly evolve to reach the most competitive, compact, and accurate network architecture. When evaluating an individual, we use a relatively small subset of the original training dataset to fine-tune the compressed network, which quickly excavates its potential performance. In addition, there are billions of parameters in a well designed deep network, but the number of its filters is usually only about 10k (*e.g.* AlexNet and ResNet). Therefore, the overall running times and the space complexity of the evolutionary algorithm for compressing CNNs are acceptable. Experiments conducted on several benchmark CNNs demonstrate that compressed networks are more lightweight but have comparable accuracies to their original counterparts. Beyond conventional network redundancies, we suggest that convolutional filters with either large or small weights possess redundancies, the discrepancy between filters is appreciated, and that high-frequency coefficients of convolution filters are unnecessary (*i.e.* smooth filters are adequate).

## 2 RELATED WORKS

Our goal is to discard redundant weights and parameters in pre-trained CNNs to further reduce their online memory usage and FLOPs (floating number operations, which is mainly accumulated by floating number multiplications). There is a number of works proposed for discovering redundancy in pre-trained CNN models, which can be divided into two categories based on techniques they used.

### 2.1 Parameter Refining

There are usually billions of parameters in a well designed neural networks [15, 36], and the most intuitive method is to excavate which of them are redundant for the entire inference. Thus, lots of approaches were proposed for refining parameters in convolution filters in CNNs.

**Parameter decomposition.** Denton *et al.* [8] used singular value decomposition (SVD) technique to discover the low-rank approximation of parameters in the fully-connected layer. Kim *et al.* [22] utilized tensor decomposition techniques to speed up fully-connected layers. Lebedev *et al.* [25] used rank-1 filters to reduce the number of parameters. In addition, since there are similar parameters in different convolution filters, Gong *et al.* [12] employed k-means to obtain the cluster centers of weights of convolution filters, and then approximately represented convolution filters using their corresponding clustering centers. Chen *et al.* [4] used a hash function to randomly cluster weights of convolution filters, so that weights belonging to the same hash bucket can be represented using a single parameter. Jaderberg *et al.* [21] constructed a small dictionary using a set of pre-learned bases to represent convolution filters. If the number of bases is much less than the dimensionality of convolution filters, the storage complexity will be reduced significantly.

**Parameter quantization.** Parameters in traditional CNNs were trained using 32-bit floating values, which is fine-drawn. Therefore, Vanhoucke *et al.* [39] proposed a fixed-point implementation with 8-bit integer values, and Hwang and Sung [20] utilized 3-bit values to represent parameters. Furthermore, a lot of binarization works [5, 6, 31, 41] were proposed to reduce the parameter redundancy of CNNs, thus calculations and storages of 32-bit values were significantly reduced to those of binary values. However, the parameter binarization strategy will sacrifice too much accuracy of the original network. Hence, Arora *et al.* [1] utilized +1/0/-1 parameters to relax the +1/-1 constraint for enhancing the performance of the binaryzied network.

**Parameter pruning.** Since the convolution operation can be seen as weighted combination of input data and parameter in convolution filters, subtle parameters tends to have limited influence on the output results. Therefore, Han *et al.* [14] demonstrated that most of parameters could be removed without affecting the performance of the pre-trained network, and [13] further compactly stored the generated sparse networks in sparse row format (CSR) [3] and Huffman encoding [19]. Wang *et al.* [44] expanded the pruning approach to the DCT frequency domain to achieve higher compression and speed-up ratios. On the other side, dictionary learning based methods can also be introduced for pruning useless parameters. For example, Liu *et al.* [27] learned a set of kernel bases, and then transferred original filters in the coefficient domain with high sparsity, Bagherinezhad *et al.* [2] decomposed original convolution filters as weighted combinations of basis filters and sparse coefficients.

### 2.2 Architecture Refining

In addition to these parameter refining methods which change the basic convolution component in CNNs, there are lots of works focusing on refining architectures of pre-trained neural networks.

Hinton *et al.* [17] first proposed the teacher-student paradigm for learning a thinner neural network by minimizing the difference between the compressed network and the teacher network. Romero *et al.* [34] further explored a method for inheriting information from the teacher network at an arbitrary layer, which enhances the accuracy of the thinner and deeper student network. McClure and Kriegeskorte [29] minimized the pairwise distance of samples between the compressed network and the original network. Wang *et al.* [43] proposed to exploit adversarial loss to supervise the compression. You *et al.* [47] utilized multiple teacher networks for training the compressed network. [50] introduced the attention loss for enhancing the student network. Wang *et al.* [42] exploited a compact feature map learning methods and established portable networks. *etc.* However, performance of the student network is usually lower than that of the teacher network, and all pre-trained convolution filters in original CNNs were abandoned, which are useful for maintaining the performance of compressed networks.

Moreover, some works expanded the parameter pruning to channel pruning or neuron pruning, which also produces compressed networks with more efficient architectures. For example, Hu *et al.* [18] proposed to remove neurons in fully connected layers with subtle outputs. Wen *et al.* [46] excavated redundancy by pruning weights in different aspects (*e.g.* channels, filters, neurons). He *et al.* [16] explored a LASSO regression based channel selection for refining the architecture of a pre-trained CNN. Luo *et al.* [28] proposed to discard filters with less importances for establishing portable networks. Yu *et al.* [48] proposed to remove neurons with less scores in the propagation. However, these methods face the same problem as that of parameter pruning methods, *i.e.* the redundancy in CNNs are not completely exacted.

It is obvious that architecture refining methods are much more flexible since compressed network by utilizing these methods can be perfectly supported by any off-the-shelf deep learning libraries. In fact, compressed network with portable architectures can be further compressed by parameter refining methods such as deep compression [13] and CNNpack [44].

## 3 EVOLUTIONARY COMPRESSION

Most existing CNN compression methods are based on the consensus that weights or filters with subtle values have limited influence on the performance of the original network. In this section, we introduce an evolutionary algorithm to significantly excavate redundancy in CNNs and devise a novel compression method.

### 3.1 Modeling Filters Redundancy

Considering a convolutional neural network $\mathcal{L}$ with $p$ convolutional layers, we define $p$ sets of convolution filters $\mathbf{F} = \{F_1, ..., F_p\}$ for these layers. For the $i$-th convolutional layer, its filter is denoted as $F_i \in \mathbb{R}^{H_i \times W_i \times C_i \times N_i}$, where $H_i$ and $W_i$ are the height and width of filters, respectively, $C_i$ is the channel size, and $N_i$ is the number of filters in this layer. Given a training sample $\mathcal{X}$ and the corresponding ground truth $\mathcal{Y}$, the error of the network can be defined as $E(\mathcal{X}, \mathcal{Y}, \mathbf{F})$, which could be, for example, softmax or Euclidean losses. The conventional parameter pruning algorithm [13, 14] can be formulated as

$$\begin{aligned} \min_{B_1,...,B_p} \; & ||E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}}) - \mathcal{Y}||_F^2 + \lambda \sum_{i=1}^{p} ||B_i||_1, \\ s.t. \quad & \hat{\mathbf{F}} = \{F_1 \circ B_1, ..., F_p \circ B_p\}, \\ & B_i \in \{0,1\}^{H_i \times W_i \times C_i \times N_i}, \; \forall \; i = 1, ..., p, \end{aligned} \tag{1}$$

where $B_i$ is a binary tensor for removing redundant weights in $\mathbf{F}$, $|| \cdot ||_1$ is the $\ell_1$-norm accumulating absolute values $B_i$, *i.e.* the number in $B_i$, $\circ$ is the element-wise product, $|| \cdot ||_F$ is the Frobenius norm for matrices, and $\lambda$ is the tradeoff parameter. A larger $\lambda$ will make $B_i$ more sparse and so a network parameterized with $\mathbf{F}$ will have fewer weights.

In general, Fcn. 1 is easy to solve if $E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}})$ is a linear mapping of $\hat{\mathbf{F}}$. However, neural networks are composed of a series of complex operations, such as pooling and ReLU, which increase the complexity of Fcn. 1. Therefore, a greedy strategy [13, 46] has been introduced to obtain a feasible solution that removes weights with small absolute values:

$$B_i^{(h,w,c,n)} = \begin{cases} 0, & \text{if } |F_i^{(h,w,c,n)}| \leq \tau, \\ 1, & \text{otherwise}, \end{cases} \tag{2}$$

where $\tau > 0$ is a threshold. This strategy is based on the intuitive assumption that small weights make subtle contributions to the calculation of the convolution response.

Although sparse filters learned by Fcn. 1 demand less storage and computational resources, the size of the feature maps produced by these filters does not change. For example, a convolutional layer with 10 filters will generate 10 feature maps for one input data before and after compression, which accounts for a large proportion of online memory usage. Moreover, these sparse filters usually need some additional techniques to support and speed-up their compression such as CuSparse kernel, CSR format, or the fixed-point multiplier [13]. Therefore, more flexible approaches [10, 18, 46] have been developed to directly discard redundant filters in a given convolutional layer:

$$B_i^{(:,:,:,n)} = \begin{cases} 0, & \text{if } ||F_i^{(:,:,:,n)}||_F^2 \leq \tau, \\ 1, & \text{otherwise}, \end{cases} \tag{3}$$

where $B_i^{(:,:,:,n)}$ denotes the $n$-th filter in the $i$-th convolutional layer. By directly removing convolution filters, network complexity can be significantly decreased. However, Fcn. 3 is also biased since the Frobenius norm of filters is not a reasonable redundancy indicator. For example, most of the weights in a filter for extracting edge information are very small. Thus, a more accurate approach for identifying redundancy in CNNs is urgently required.

### 3.2 Modeling Redundancy in CNNs by Exploiting Evolutionary Algorithms

Instead of the greedy strategies shown in Fcns. 2 and 3, evolutionary algorithms such as the genetic algorithm (GA [7, 32]) and simulated annealing (SA [23]) have been widely applied to the NP-hard binary programming problem. A series of bit (0 or 1) strings (*individuals*) are used to represent possible solutions of the binary programming problem, and these individuals evolve using some pre-designed operations to maximize their *fitnesses*.

A binary variable can be attached to each weight in the CNN $\mathcal{L}$ to indicate whether the weight takes effect or not, but a large number of binary variables will significantly slow down the CNN compression process, especially for sophisticated CNNs learned over large-scale datasets (*e.g.* ImageNet [35]). For instance, AlexNet [24] has eight convolutional layers with more than $6 \times 10^7$ 32-floating weights in total, so it is infeasible to generate a population with hundreds of $6 \times 10^7$-dimensional individuals. In addition, as mentioned above, excavating redundancy in convolution filters itself produces a regular CNN model with less computational complexity and memory usage, which is more suitable for practical applications. Therefore, we propose to assign a binary bit to each convolution filter in a CNN, and these binary bits form an *individual* $\mathbf{b}$ for this network. By doing so, the dimensionality of $\mathbf{b}$ is tolerable, *e.g.* $\mathbf{b} \in \{0,1\}^{9568}$ (without the last 1000 convolution filters corresponding to the 1000 classes in the ILSVRC 2012 dataset) for AlexNet.

During evolution, we use GA to constantly update individuals of greater fitness. Other evolutionary algorithms can be applied using a similar approach. The compression task has two objectives: preserving performance and removing the redundancy of the original networks. The fitness of a specific *individual* $\mathbf{b}$ can therefore be defined as

$$f(\mathbf{b}) = 1 - E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}}) + \frac{\lambda}{L} \sum_{i=1}^{p} ||1 - \mathbf{b}_i||_1, \tag{4}$$

where $\mathbf{b}_i$ denotes the binary bit for the $i$-th convolution filter in the given network, and $L$ is the number of all convolution filters in the network. $E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}})$ calculates the classification loss of the network using compressed filters $\hat{\mathbf{F}} = \{F_1 \circ B_1, ..., F_p \circ B_p\}$, which supposed as a general loss taken value from 0 to 1. In addition, we include a constant 1 in Fcn. 4, which ensures $f(\mathbf{b}) > 0$ for the convenience of calculating the probability of each individual in the evolutionary algorithm process. $\lambda > 0$ is the tradeoff parameter, and

$$B_i^{(:,:,:,n)} = \begin{cases} 0, & \text{if } \mathbf{b}_i(n) = 0, \\ 1, & \text{otherwise}, \end{cases} \tag{5}$$

where $\mathbf{b}_i(n) = 0$ implies that the $n$-th filter in the $i$-th layer has been discarded, otherwise retained.

In addition, the last term in Fcn. 4 implicitly assumes that discarding every convolution filter makes an equivalent contribution to compression. However, the memory utilization of filters in different convolutional layers is different and related to the height $H$, width $W$, and the number of channel $C$. Therefore, filter size must be taken into account, and Fcn. 4 can be reformulated as:

$$f(\mathbf{b}) = 1 - E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}}) + \frac{\lambda}{M} \sum_{i=1}^{p} (H_i W_i C_i \cdot ||1 - \mathbf{b}_i||_1), \quad (6)$$

where $H_i$, $W_i$, $C_i$ are height, width, and channel number of filters in the $i$-th convolutional layer, respectively. $M = \sum_{i=1}^{p} H_i W_i C_i N_i$ is the total number of weights in the network, which scales the last term in Fcn. 6 to $[0, \lambda]$.

In addition, the number of channels $C_i$ in the $i$-th layer is usually set as the number of convolution filters $N_{i-1}$ ($N_0 = 3$ for RGB color images) in the $(i-1)$-th layer to make two consecutive network layers compatible. Instead of fixing $C_i$ in Fcn. 6, $C_i$ should vary with $\mathbf{b}_{i-1}$. Thus, we reformulate the calculation of fitness as follows:

$$f(\mathbf{b}) = 1 - E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}}) + \frac{\lambda}{M} \sum_{i=1}^{p} (H_i W_i \cdot ||1 - \mathbf{b}_{i-1}||_1 \cdot ||1 - \mathbf{b}_i||_1), \quad (7)$$

where $\mathbf{b}_p(n) = 1, \forall\, n = 1, ..., N_p$ for the last layer consisting of nodes corresponding to different classes in a particular dataset. The second objective in Fcn. 7 accumulates the discarded weights of the compressed network. Since the error rate of a network tends to be influenced by adjusting the network architecture, we use a subset of the training data (10k images randomly extracted from the training set) to fine-tune the network weights and then recalculate $E(\mathcal{X}, \mathcal{Y}, \hat{\mathbf{F}})$ to provide a more reasonable evaluation. This fine-tuning is fast, since compressed networks with fewer filters require much less computation, *e.g.* fine-tuning over 10k images will cost about 2 seconds for LeNet and about 30 seconds for AlexNet, which is tolerable. Then, GA is deployed to discover the fittest individual through several evolutions detailed in the next section.

## 3.3 Genetic Algorithm for Compression

GA can automatically search for compact neural networks by alternately evaluating the fitness of each individual in the whole population and executing operations on individuals. The population in the current iteration are regarded as parents, who breed another population as offspring using some representative operations, including **selection**, **crossover**, and **mutation**, with the expectation that the subsequent offsprings are fitter than the preceding parents. First, each individual is given a probability by comparing its fitness against those of other *individuals* in the current *population*:

$$\Pr(\mathbf{b}^j) = f(\mathbf{b}^j) \Big/ \sum_{k=1}^{K} f(\mathbf{b}^k), \quad (8)$$

where $K$ is the number of *individuals* in the *population*. Then, the above three operations will be randomly applied as follows:

**Selection.** Given a probability parameter $s_1$, an individual is selected according to Fcn. 8 and then directly duplicated as an offspring. It is clear that compressed networks with higher accuracy and compression ratios will be preserved. The best individual in the parent population is usually inherited to preserve the optimal solution.

**Algorithm 1** Evolution method for compressing CNNs.

```
Input: An image dataset $\{\mathcal{X}, \mathcal{Y}\}$ including $n$ images for evaluating
    individuals, a pre-trained convolutional neural network $\mathcal{L}$, parameters:
    the scale of the population $K$, the maximum iteration number $T$, $\lambda$,
    $s_1$, $s_2$, and $s_3 = 1 - s_1 - s_2$.
 1: Randomly initialize the population $P_1$, each individual is represented as a
    binary vector w.r.t. convolution filters in the given network $\mathcal{L}$;
 2: for $t = 2$ to $T$ do
 3:   Calculate the fitness of individuals in $P_{t-1}$ (Fcn. 7);
 4:   Calculate probability of selecting each individual Fcn. 8;
 5:   for $k = 2$ to $K$ do
 6:     $P_t^{(1)} \leftarrow$ the best individual in $P_{t-1}$;
 7:     Generate a random value $s \sim [0, 1]$;
 8:     if $s < s_1$ then
 9:       $P_t^{(k)} \leftarrow$ a randomly selected parent (Fcn. 8);
10:     else if $s_1 \leq s < s_1 + s_2$ then
11:       Randomly select two parents;
12:       Generate two offspring (Fcn. 9) and calculate their fitnesses according to Fcn. 7;
13:       $P_t^{(k)} \leftarrow$ the best offspring;
14:     else
15:       $P_t^{(k)} \leftarrow$ a randomly selected parent after applying XOR
          on a fragment according to Fcn. 10;
16:     end if
17:   end for
18: end for
19: Use the optimal individual in $P_T$ to construct a compact neural network
    $\hat{\mathcal{L}}$, and maintain convolution filters in $\mathcal{L}$;
Output: The compressed $\hat{\mathcal{L}}$ after fine-tuning.
```

**Crossover.** Given a probability parameter $s_2$, two selected parents according to Fcn. 8 will be crossed to generate two offspring as follows:

$$\begin{array}{ll} \text{parent}_1 : 01010\Big|\mathbf{1110010}\Big|0101 & \text{parent}_2 : 01011\Big|\mathbf{0101011}\Big|0110 \\ \text{offspring}_1 : 01010\Big|\mathbf{0101011}\Big|0101 & \text{offspring}_2 : 01011\Big|\mathbf{1110010}\Big|0110 \end{array} \quad (9)$$

The objective of the crossover operation is to integrate excellent information from the parents. The fitness of two offspring are different, and we discard the weaker one.

**Mutation.** To promote population diversity, mutation randomly changes a fragment in the parent and produces an offspring. The conventional mutation operation for binary encoding is a XOR operation as follows:

$$\text{parent} : 100\Big|\mathbf{10010101}\Big|101010 \quad \text{offspring} : 100\Big|\mathbf{01101010}\Big|101010 \quad (10)$$

The parent is also selected according to Fcn. 8, and the mutation operation is performed with a probability parameter $s_3$. Since the scale of offspring (*i.e.* the number of binary codes $K$) is the same as that of parents, we have $s_1 + s_2 + s_3 = 1$.

By iteratively employing these three genetic operations, the initial population will be updated efficiently until the maximum iteration number is achieved. After obtaining the individual with optimal fitness, we can reconstruct a compact CNN. Then, the fine-tuning strategy is adopted to enhance the performance of the

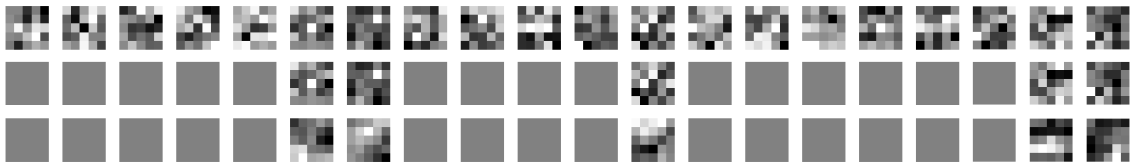


**Figure 2: Convolution filters learned on MNIST. From top to bottom: the original convolution filters, filters after applying the proposed method, and filters after fine-tuning.**

**Table 1: Compression statistics for AlexNet.**

| Layer | Num of Weights | Memory | Num of (New) Weights | Memory | $r_c$ |
|---|---|---|---|---|---|
| conv1 | $11 \times 11 \times 3 \times 96$ | 1.24MB | $11 \times 11 \times 3 \times 56$ | 0.08MB | 1.71× |
| conv2 | $5 \times 5 \times 48 \times 256$ | 1.88MB | $5 \times 5 \times 28 \times 120$ | 0.32MB | 3.66× |
| conv3 | $3 \times 3 \times 256 \times 384$ | 3.62MB | $3 \times 3 \times 120 \times 190$ | 0.78MB | 4.31× |
| conv4 | $3 \times 3 \times 192 \times 384$ | 2.78MB | $3 \times 3 \times 95 \times 188$ | 0.61MB | 4.12× |
| conv5 | $3 \times 3 \times 94 \times 144$ | 1.85MB | $3 \times 3 \times 175 \times 226$ | 0.46MB | 3.63× |
| fc6 | $6 \times 6 \times 256 \times 4096$ | 144MB | $6 \times 6 \times 144 \times 1386$ | 27.41MB | 5.25× |
| fc7 | $1 \times 1 \times 4096 \times 4096$ | 64MB | $1 \times 1 \times 1386 \times 1848$ | 9.77MB | 6.55× |
| fc8 | $1 \times 1 \times 4096 \times 1000$ | 15.62MB | $1 \times 1 \times 1848 \times 1000$ | 7.05MB | 2.22× |
| Total | 60954656 | 232.52MB | 12186444 | 46.48MB | 5.00× |

compressed network. Alg. 1 summarizes the proposed evolutionary method for compression.

## 4 COMPRESSION AND SPEED-UP IMPROVEMENTS

In the above section, we presented the evolutionary method for compressing pre-trained CNN models. Since there is at least one convolution filter per layer in the compressed network $\hat{\mathcal{L}}$, it has the same depth but less filters in $\hat{\mathbf{F}}$ compared to the original network $\mathcal{L}$ with $\mathbf{F}$. Here we further analyze the memory usage and computational cost of compressed CNNs using Alg. 1.

**Speed-up ratio.** For a given image, the $i$-th convolutional layer $\mathcal{L}_i$ produces feature maps $Y_i \in \mathbb{R}^{H'_i \times W'_i \times N_i}$ through a set of convolution filters $F \in \mathbb{R}^{H_i \times W_i \times C_i \times N_i}$, where $H'_i$ and $W'_i$ are the height and width of feature maps, respectively, and $C_i = N_{i-1}$. Since multiplications of 32-bit floating values are much more expensive than additions, and there is more computation in other auxiliary layers (*e.g.* pooling, ReLU, and batch normalization), speed-up ratios are usually calculated on these floating number multiplications [31, 44]. Considering the major computational cost, the speed-up ratio of the compressed network for this layer compared to the original network is

$$r_{s_i} = \frac{H_i W_i N_{i-1} N_i H'_i W'_i}{H_i W_i \hat{N}_{i-1} \hat{N}_i H'_i W'_i} = \frac{N_{i-1} N_i}{\hat{N}_{i-1} \hat{N}_i}, \tag{11}$$

where $\hat{N}_i = ||1 - \mathbf{b}_i||_1$ is the number of filters in the $i$-th convolutional layer of the compressed network, as shown in Fcn. 7. Besides the filter number $\hat{N}_i$ of a layer, $\hat{N}_{i-1}$ also has a greater impact on $r_{s_i}$, suggesting that it is very difficult to directly find an optimal compact architecture of the original network. Moreover, excavating redundancy in the filter itself may be a more promising way to speed it up, *e.g.* if we discard half of the filters per layer, the speed-up ratio of the proposed method is about 4×.

**Compression ratio.** The compression ratio on convolution filters is easy to calculate and is equal to the last term in Fcn. 7. Specifically, for the $i$-th convolutional layer, the compression ratio of the proposed method is

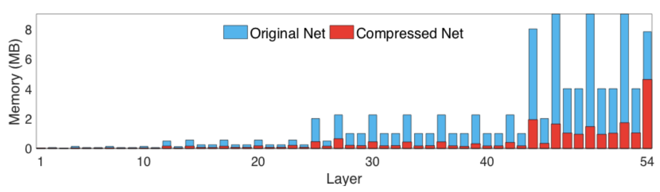
$$r_{c_i} = \frac{H_i W_i N_{i-1} N_i}{H_i W_i \hat{N}_{i-1} \hat{N}_i} = \frac{N_{i-1} N_i}{\hat{N}_{i-1} \hat{N}_i}. \tag{12}$$

Besides the convolution filters, there are other memory usages that are often ignored in existing parameter refining methods such as deep compression [13] and CNNpack [44] *etc.* Although FLOPs of the compression network are significantly reduced after applying these methods, the network will produce the same amount of feature maps though there is only one parameter in each convolution filter.

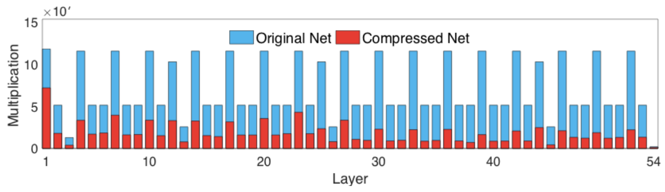
In fact, the feature maps of different layers account for a large proportion of online memory. In some implementations [40], the feature maps of a layer are removed after they have been used to calculate the following layer to reduce the online memory usage. However, memory allocation and release are time consuming. In addition, short-cut layers are widely used in recent CNN models [15], in which previous feature maps are preserved for combination with other layers. Discarding redundant convolutional filters significantly reduces the memory usage of feature maps. For a given convolutional layer $\mathcal{L}_i$, the compression ratio of the proposed method on feature maps is

$$r_{f_i} = \frac{N_i H'_i W'_i}{\hat{N}_i H'_i W'_i} = \frac{N_i}{\hat{N}_i}, \tag{13}$$

which is directly affected by the sparsity of $\mathbf{b}_i$. Accordingly, the memory to store the feature maps of other layers (*e.g.* pooling and ReLU) will be reduced simultaneously. The experimental results



(a) Compression ratio of all convolution filters ($r_c$)



(b) Speed-up ratio of all convolutional layers ($r_s$)

**Figure 3: Compression statistics for ResNet-50.**

**Table 2: Compression statistics for VGG-16 Net.**

| Layer | Num of Weights | Memory | Num of (New) Weights | Memory | $r_c$ |
|---|---|---|---|---|---|
| conv1_1 | $3 \times 3 \times 3 \times 64$ | 12.26MB | $3 \times 3 \times 3 \times 12$ | 0.001MB | 5.33× |
| conv1_2 | $3 \times 3 \times 64 \times 64$ | 12.39MB | $3 \times 3 \times 12 \times 28$ | 0.01MB | 12.19× |
| conv2_1 | $3 \times 3 \times 64 \times 128$ | 6.41MB | $3 \times 3 \times 28 \times 57$ | 0.05MB | 5.13× |
| conv2_2 | $3 \times 3 \times 128 \times 128$ | 6.69MB | $3 \times 3 \times 57 \times 61$ | 0.12MB | 4.71× |
| conv3_1 | $3 \times 3 \times 128 \times 256$ | 4.19MB | $3 \times 3 \times 61 \times 133$ | 0.28MB | 4.04× |
| conv3_2 | $3 \times 3 \times 256 \times 256$ | 5.31MB | $3 \times 3 \times 133 \times 127$ | 0.58MB | 3.88× |
| conv3_3 | $3 \times 3 \times 256 \times 512$ | 5.31MB | $3 \times 3 \times 127 \times 137$ | 0.60MB | 3.77× |
| conv4_1 | $3 \times 3 \times 512 \times 512$ | 6.03MB | $3 \times 3 \times 137 \times 194$ | 0.91MB | 4.93× |
| conv4_2 | $3 \times 3 \times 512 \times 512$ | 10.53MB | $3 \times 3 \times 194 \times 119$ | 0.79MB | 11.36× |
| conv4_3 | $3 \times 3 \times 512 \times 512$ | 10.53MB | $3 \times 3 \times 119 \times 320$ | 1.31MB | 6.88× |
| conv5_1 | $3 \times 3 \times 512 \times 512$ | 9.38MB | $3 \times 3 \times 320 \times 72$ | 0.79MB | 11.38× |
| conv5_2 | $3 \times 3 \times 512 \times 512$ | 9.38MB | $3 \times 3 \times 72 \times 62$ | 0.15MB | 58.72× |
| conv5_3 | $3 \times 3 \times 512 \times 512$ | 9.38MB | $3 \times 3 \times 62 \times 122$ | 0.26MB | 34.66× |
| fc6 | $7 \times 7 \times 512 \times 4096$ | 392MB | $7 \times 7 \times 122 \times 2300$ | 52.45MB | 7.47× |
| fc7 | $1 \times 1 \times 4096 \times 4096$ | 64MB | $1 \times 1 \times 2300 \times 125$ | 1.09MB | 58.36× |
| fc8 | $1 \times 1 \times 4096 \times 1000$ | 15.62MB | $1 \times 1 \times 125 \times 1000$ | 0.48MB | 32.77× |
| Total | 138344128 | 579.46MB | 20118610 | 59.88MB | 8.81× |

and a discussion of compression and speed-up ratios are presented in the following section.

## 5 EXPERIMENTS

**Baseline methods and Datasets.** The proposed method was evaluated on four baseline CNN models: LeNet [26], AlexNet [24], VGGNet-16 [36], and ResNet-50 [15], and conducted using the MNIST handwritten digit and ILSVRC datasets. We used MatConvNet [40] and NVIDIA Titan X graphics cards to implement the proposed method. In addition, several state-of-the-art approaches were selected for comparison including both parameter refining (*e.g.* deep compression [13], CNNpack [44]) and architecture refining algorithms (*e.g.* Trimming [18] and ThiNet [28]).

**LeNet on MNIST.** The performance of the proposed method was first evaluated on a small network to study some of its characteristics. The network has four convolutional layers of size $5 \times 5 \times 1 \times 20$, $5 \times 5 \times 20 \times 50$, $4 \times 4 \times 50 \times 500$, and $1 \times 1 \times 500 \times 10$, respectively. The model was trained with batch normalization layers and the accuracy was 99.20%.

The proposed method has several parameters as shown in Alg. 1. Population $K$ was set to 1000 to ensure a sufficiently large search space, and the maximum iteration number $T$ was set to 100. Three probability parameters were empirically set to $s_1 = 0.2$, $s_2 = 0.7$, and $s_3 = 0.1$ [7]. A larger $\lambda$ makes the compressed network more compact, but the accuracy of the original network cannot be retained in the compressed counterpart. We tuned this parameter from 0.5 to 1.5 and set it to 0.9, which was the best trade-off between network accuracy and compression ratio, since the accuracy of the original network was very high and each individual could maintain considerable accuracy.

The evolutionary process for compressing the network is shown in Fig. 1. Individuals in the population are updated with higher fitness individuals after each iteration. As a result, we obtained a $103KB$ compressed network that consistutes of four convolutional layers: $5 \times 5 \times 1 \times 9$, $5 \times 5 \times 9 \times 17$, $4 \times 4 \times 17 \times 84$, and $1 \times 1 \times 84 \times 10$, respectively. The model accuracy after fine-tuning was 99.20%, *i.e.* there was no decrease in accuracy. Compression and speed-up ratios of the entire network were $r_c = 15.52\times$, $r_s = 5.76$, and $r_f = 2.42$.

Furthermore, for fair comparison, we directly initialized a network with the same architecture (*i.e.* we directly used Gaussian random values to initialize a network of the same architecture as that of the resulting network from the proposed ECS) and tuned it on MNIST. Unfortunately, the accuracy of this network was only 98.5%, significantly lower than that of the original network since it cannot inherit pre-trained convolution filters of the original network. Moreover, given the same total number of filters as that of the resulting network from the proposed ECS, we designed an ordinary network by randomly choosing the number of filters in each layer.

**Table 3: An overall comparison between state-of-the-art CNN compression methods and the proposed evolutionary compression scheme (ECS) on the ILSVRC2012 dataset. The overall compression and speed-up ratios are denoted $r_c$ and $r_s$, respectively.**

| Model | Evaluation | Original | Trim [18] | ThiNet [28] | ECS | DeepComp [13] | CNNpack [44] | ECS+CNNpack |
|---|---|---|---|---|---|---|---|---|
| AlexNet [24] | $r_c$ | 1 | - | - | 5.00× | 35× | 39× | 44.83× |
| | $r_s$ | 1 | - | - | 3.34× | - | 25× | 26.61× |
| | *top*-1 *err* | 41.8% | - | - | 41.9% | 42.7% | 41.6% | 41.9% |
| | *top*-5 *err* | 19.2% | - | - | 19.2% | 19.7% | 19.2% | 19.3% |
| VGGNet [36] | $r_c$ | 1 | 2.70× | 16.63× | 8.81× | 49× | 46× | 50.19× |
| | $r_s$ | 1 | ≈1× | 3.31× | 5.88× | 3.5× | 9.4× | 10.24× |
| | *top*-1 *err* | 28.5% | 26.7% | 32.6% | 29.5% | 31.1% | 29.7% | 29.5% |
| | *top*-5 *err* | 9.9% | 8.7% | 12.1% | 10.2% | 10.9% | 10.4% | 10.3% |
| ResNet [15] | $r_c$ | 1 | - | 2.95× | 4.10× | - | 12.2× | 14.64× |
| | $r_s$ | 1 | - | 3.5× | 3.83× | - | 4× | 5.19× |
| | *top*-1 *err* | 24.6% | - | 31.6% | 25.3% | - | - | 25.2% |
| | *top*-5 *err* | 7.7% | - | 11.7% | 8.1% | - | 7.8% | 7.9% |

The network accuracy was about 92.7%, demonstrating that the proposed method provides an effective architecture for constructing a portable network and inherits useful information from the original network.

**Filter visualization.** The proposed method excavates redundant convolution filters using an evolutionary algorithm, which is different to the existing weight or filter pruning approaches. Therefore, it is necessary to explore which filters are recognized as redundant and which convolution filters are optimal for CNNs. To this end, we visualized the LeNet filters on MNIST before and after applying the proposed method, as shown in Fig. 2.

The result shown in the second row of Fig. 2 is particularly interesting. Our method not only discards small filters but also removes some filters with large weights. Of note, the remaining filters after fine-tuning are even more distinct. The average Euclidean distance of filters in the third row (compressed network) is 0.2428, while Euclidean distances of filters in the first and second rows are 0.1927 and 0.1789, respectively. This observation shows that redundancy can exist in either large or small convolution filters, and similar filters may be redundant and non-contributory to the entire CNN, providing a strong a priori rationale for designing and learning CNN architectures. In addition, the filters shown in the third line are obviously smoother than those in the original network, demonstrating the feasibility of compressing CNNs in the frequency domain as discussed in [44].

**Compressing convolutional networks on ImageNet.** We next employed the proposed Evolutionary Compression Scheme (ECS) on ImageNet ILSVRC 2012 [35], which contains 1.2 millions images for training and 50k images for testing. We examined three mainstream CNN architectures: AlexNet [24], VGGNet-16 [36], and ResNet-50 [15]. There are over 61M parameters in AlexNet and over 138M weights in VGGNet-16. ResNet-50 has about 25M parameters, which is more compact than the previous two CNNs. The top-5 accuracies of these three models were 80.8%, 90.1%, 92.9%, respectively.

Since the accuracy of convolutional networks on ImageNet was much harder to maintain, we adjusted $\lambda = 0.5$ to allow individuals higher accuracy evolution. The compression and speed-up ratios of the proposed methods on the three CNNs are shown in Table 3. In addition, architectures of compressed AlexNet and VGGNet-16 are shown in Tab. 1 and Tab. 2, and detailed compression and speed-up results of ResNet-50 are shown in Fig. 3.

**Compared with architecture refining methods.** A detailed comparison of the above three benchmark deep CNN models between the proposed method and state-of-the-art methods can be found in Table 3. We first compared the proposed ECS with two architecture refining approaches, *i.e.* Trimming [18] and ThiNet [28]. Trimming only compresses fully connected layers, whose memory usage accounts for the largest proportion in VGGNet while calculation accounts for a very small proportion of the entire network. Therefore, its speed-up ratio is approximate to 1 and it achieved a 2.7× compression ratio on the VGGNet. ThiNet firstly replaces fully connected layers by global average pooling (GAP) layers to eliminate over 80% parameters with an acceptable accuracy decline, and then removes useless filters in convolutional layers to obtain a very high compression ratio on the VGGNet. The ResNet-50 has a more compact architecture, whose layers cannot be directly replaced by GAP layers. Thus, the compression ratio of ThiNet is only 2.95×, and the speed-up ratio is 3.5×. Contrastively, we excavated redundant convolution filters globally yielding higher compression performance, and accuracies of compressed networks by exploiting ECS are higher than those of ThiNet.

**Compared with parameter refining methods.** As mentioned above, compressed networks by exploiting architecture refining methods are still regular CNNs, which can be further squeezed by exploiting parameter refining approaches such as sparse shrinkage and Huffman encoding. Therefore, we further applied CNNpack [44] on models generated by ECS as detailed in Table 3. Results of the proposed method are slightly higher than those of CNNpack, since networks we used before pruning, quantization, and encoding are more portable than those used in the original CNNpack. It is obvious that ECS+CNNpack can surpass all comparison methods since we eliminate redundant parameters and filters as much as possible.

However, neural networks compressed with parameter refining techniques such as sparsity and Huffman encoding cannot be easily launched on existing libraries, since convolution operations in these compressed network cannot be directly implemented using original codes. Differently, compressed CNNs using ECS can be directly utilized and deployed on any platform, which are more flexible.

**Compression ratios on feature maps.** As discussed in Fcn. 13, the compression ratio on CNN feature maps is also an important metric for evaluating compression methods (since we need to save feature maps of intermedia layers for subsequent calculating in recent CNN models such as ResNets [15]), but it is ignored in most existing parameter refining approaches. Therefore, the compression ratios on feature maps of these methods are both equal to 1× such as [8, 13, 44].

**Table 4: Compression ratios of CNN feature maps.**

| Model | AlexNet | VGGNet-16 | ResNet-50 |
|---|---|---|---|
| Original memory | 5.49 *MB* | 109.26 *MB* | 137.25 *MB* |
| Compression ratio $r_f$ | 1.88× | 2.54× | 1.86× |

The compression ratios on feature maps of the proposed method on different CNNs are shown in Table 4. Note that other architecture refining methods can also achieve considerable feature map compression ratios, but they did not report these results. However, the amount of feature maps indicate the required calculation. Since the proposed method obtained higher speed-up ratios, its $r_f$ values are definitely higher than those of other methods such as Trimming and ThiNet. Overall, it is clear that the compressed networks using ECS are more portable which can not only reduce memory usage and computational complexity produced by convolution filters, but also save the memory usage brought by considerable feature maps. Therefore, the proposed method can provide more online benefits than other compression methods.

## 6 CONCLUSION

CNNs with higher performance and portable architectures are urgently required for mobile devices. This paper presents an effective CNN compression technique using an evolutionary algorithm. Compared to state-of-the-art methods, we no longer directly recognize some weights or filters as redundant according to some priori, *e.g.* subtle weights or filters. The proposed method identifies redundant convolution filters by iteratively refining a certain number of networks before learning a compressed network with significantly fewer parameters. Our experiments show that the proposed method can achieve significant compression and speed-up ratios and retain the classification accuracy of the original neural network. Moreover, the network compressed by the proposed approach is still a regular CNN that can be directly used for online inference without any decoding or other sophisticated technical supports.

## ACKNOWLEDGMENTS

This work was supported in part by the Australian Research Council under Projects: FL-170100117, DE-180101438, DP-180103424, DP-140102164, and LP-150100671. We also thank supports of NSFC 61375026 and 2015BAF15B00.

## REFERENCES

[1] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. 2014. Provable bounds for learning some deep representations. *ICML* (2014).

[2] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. 2017. LCNN: Lookup-based Convolutional Neural Network. In *CVPR*.

[3] Nathan Bell and Michael Garland. 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*.

[4] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *ICML*.

[5] Matthieu Courbariaux and Yoshua Bengio. 2016. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre Binaryconnect David. 2015. Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363* (2015).

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[8] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*.

[9] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2016. Image super-resolution using deep convolutional networks. *IEEE TPAMI* 38, 2 (2016), 295–307.

[10] Michael Figurnov, Dmitry Vetrov, and Pushmeet Kohli. 2016. Perforatedcnns: Acceleration through elimination of redundant convolutions. *NIPS* (2016).

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

[12] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[13] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[14] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *NIPS*.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[16] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *ICCV*.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[18] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv preprint arXiv:1607.03250* (2016).

[19] David A Huffman and others. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

[20] Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *IEEE Workshop on Signal Processing Systems*.

[21] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *BMVC*.

[22] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*.

[23] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, and others. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

[25] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2015. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[27] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *CVPR*.

[28] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*.

[29] Patrick McClure and Nikolaus Kriegeskorte. 2016. Representational Distance Learning for Deep Neural Networks. *Frontiers in computational neuroscience* 10 (2016).

[30] Zhou Phou, Hou Yunqing, and Feng Jiashi. 2018. Deep Adversarial Subspace Clustering. In *CVPR*.

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv preprint arXiv:1603.05279* (2016).

[32] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. 2017. Large-Scale Evolution of Image Classifiers. *arXiv preprint arXiv:1703.01041* (2017).

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*.

[34] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and others. 2015. Imagenet large scale visual recognition challenge. *IJCV* 115, 3 (2015), 211–252.

[36] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *ICLR* (2015).

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *CVPR*.

[38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *CVPR*.

[39] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*.

[40] Andrea Vedaldi and Karel Lenc. 2015. MatConvNet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*.

[41] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *ICML*.

[42] Yunhe Wang, Chang Xu, Dacheng Tao, and Chao Xu. 2017. Beyond Filters: Compact Feature Map for Portable Deep Model. In *ICML*.

[43] Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. 2018. Adversarial Learning of Portable Student Networks. In *AAAI*.

[44] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. 2016. CNNpack: Packing Convolutional Neural Networks in the Frequency Domain. In *NIPS*.

[45] Zhenyang Wang, Zhidong Deng, and Shiyao Wang. 2016. Accelerating Convolutional Neural Networks with Dominant Convolutional Kernel and Knowledge Pre-regression. In *ECCV*.

[46] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. In *NIPS*.

[47] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. 2017. Learning from Multiple Teacher Networks. In *ACM SIGKDD*.

[48] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2017. NISP: Pruning Networks using Neuron Importance Score Propagation. *arXiv preprint arXiv:1711.05908* (2017).

[49] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *CVPR*.

[50] Sergey Zagoruyko and Nikos Komodakis. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* (2016).