# Learning Filter Basis for Convolutional Neural Network Compression

Yawei Li[1*], Shuhang Gu[1*], Luc Van Gool[1,2], Radu Timofte[1]
[1]Computer Vision Lab, ETH Zurich, Switzerland, [2]KU Leuven, Belgium
{yawei.li, shuhang.gu, vangool, radu.timofte}@vision.ee.ethz.ch

## Abstract

*Convolutional neural networks (CNNs) based solutions have achieved state-of-the-art performances for many computer vision tasks, including classification and super-resolution of images. Usually the success of these methods comes with a cost of millions of parameters due to stacking deep convolutional layers. Moreover, quite a large number of filters are also used for a single convolutional layer, which exaggerates the parameter burden of current methods. Thus, in this paper, we try to reduce the number of parameters of CNNs by learning a basis of the filters in convolutional layers. For the forward pass, the learned basis is used to approximate the original filters and then used as parameters for the convolutional layers. We validate our proposed solution for multiple CNN architectures on image classification and image super-resolution benchmarks and compare favorably to the existing state-of-the-art in terms of reduction of parameters and preservation of accuracy. Code is available at https://github.com/ofsoundof/learning_filter_basis.*

## 1. Introduction

Recently, deep convolutional neural network (CNN) based approaches have been setting new state-of-the-art results not only for high-level computer vision tasks such as image classification [25, 44, 15, 18], segmentation [11, 35], and object detection [11, 10, 43, 42], but also for low-level tasks such as image super-resolution (SR) [9, 22, 33, 30, 51, 32], denoising [49, 50, 12], and deburring [38, 28]. However, most of the advances are achieved at the expense of relying on deeper architectures, millions of optimization parameters and resource-intensive computations. This hampers the application of deep neural networks under resource constrained environments, *e.g.*, mobile phones.

To overcome the above mentioned problem, one research direction is to design efficient architectures. For example, in comparison with VGG [44], both ResNet [15] and DenseNet [18] reduce the number of parameters in their CNN by one magnitude of order while achieving comparable or even more accurate image classification results. Despite being more compact, there are still redundancies in those networks, making it possible for further compression.

In the meanwhile, network compression promises to alleviate the model complexity without losing much accuracy of the original network. Many network compression methods have been proposed. They mainly fall into three categories including network quantization [41, 29, 7, 53], network pruning [13, 14, 17], and filter decomposition [20, 52, 46, 45, 40]. In this paper, we focus on filter decomposition.

Filter decomposition approximates the original filter with a lightweight convolution and a linear projection. Current methods either operate directly on the channel-wise 2D $w \times h$ filters [20, 46, 45] or decompose the intact 3D $c \times w \times h$ filters [52, 40]. For those working on 2D filters, considering that the kernel size is usually small (*e.g.*, $3 \times 3$) and a couple of new parameters are introduced to represent the 2D kernel, the compression ratio in terms of reduction of parameters is not impressive [46, 45]. The other filter decomposition methods [52] consider a 3D kernel as an intact element making impossible the reduction of the number of input channels [17]. This prevents the application of the method to narrow networks with much fewer output channels but more input channels. For example, in DenseNet-12-40, there are only 12 output channels which makes it not economic to decompose the 3D filters.

The aforementioned methods either collapse or maintain the 3D filters during decomposition, which can be regarded as 'hard' decomposition. They are only coarse-grained configurations on the two boundary operating points. The motivation of this paper is to provide the missing in-between fine-grained operating points and to balance the parameter distribution between the two decomposed convolutions. Thus, we propose a novel filter basis learning method that circumvents the limitation of the 'hard' filter decomposition methods. We split the 3D filters along the input channel dimension and each split is considered as a basic element. We assume that the ensemble of those basic elements within one convolutional layer can be represented by the linear combi-

*Equal contribution

nations of a basis. Our aim is to learn the basis and the linear combination together. During inference, the basis can be combined to reconstruct the original element, *i.e*., the 3D split. Then the splits are stacked along the input channel dimension to form the original 3D filters. Also, as it will be explained in the paper, the convolutions with respect to the original 3D filters can be converted to convolutions with respect to the learned basis. Thus, our method can be easily and efficiently implemented and embedded into the state-of-the-art networks. Compared with previous works, our basis learning method also generalized easily to $1 \times 1$ convolution, which is vital for compressing networks with intensive $1 \times 1$ convolutions.

The contribution of this paper is four-fold. **(I)** We propose a *novel basis learning method* that can reduce the input channel, making it eligible for narrow networks. Our method can be applied to convolutional layers with different kernel sizes and even $1 \times 1$ convolutions. **(II)** Our method achieves *state-of-the-art compression* performance. On VGG, SRResNet, and EDSR, our method outperforms state-of-the-art compression method gracefully with lower classification error and fewer parameters of the previous compression method [45, 46]. On ResNet, DenseNet, our compressed model is comparable with the recent state-of-the-art with fewer parameters. **(III)** Our method generalizes easily to prior work just by changing the number of splits, thus leading to *a unified formulation* of different filter decomposition methods [20, 52, 46]. **(IV)** We validate our method on both high-level vision tasks, *i.e*., image classification, and *low-level vision tasks*, *i.e*., *image SR*. Compared with the high-level vision tasks such as image classification where a single class is regressed, the low-level image SR task is more challenging since the algorithm need to recover every pixel and the content detail in the image. However, none of the previous works apply compression method to networks designed for low-level task. Our experimental results show that network compression method also works well for image SR.

The reminder of the paper is organized as follows. In Sec. 2, we discuss the relevant work. In Sec. 3, we introduce the proposed filter basis learning method for network compression in detail. In Sec. 4, we explain how to learn the basis filter and the coding coefficients. In Sec. 5, we provide the implementation details of our method and discuss the experimental results. Sec. 6 concludes the paper.

## 2. Related Work

Network compression as a research topic attracted an increased interest recently. The works in this field can be roughly grouped into three categories, namely, network pruning, network quantization, and filter decomposition.

**Network Pruning:** Network pruning attempts to prune the less important network parameters in the network. Han *et al*. [13, 14] tried to learn sparse connections and prune the less important ones. They introduced deep compression that combines several techniques such as weight pruning, quantization and weight sharing, and Huffman coding to reduce the size of neural networks. However, their method results in irregular kernel shapes making it difficult for implementation although the theoretical speed-up ratio is impressive [14]. Thus, channel pruning is proposed to remove redundant channels in feature maps which result in regular kernel shapes and implementation-friendly algorithms [3, 54, 47]. Wen *et al*. [47] explored structured sparsity including channel-wise, shape-wise, and depth-wise sparsity in deep neural networks. He *et al*. [17] proposed channel pruning to accelerate deep neural networks. Their method can choose representative channels and prune redundant ones, based on LASSO regression.

**Network Quantization:** Network quantization aims at reducing the model size of neural networks by quantizing the weight parameters. Han *et al*. [13] demonstrated how to quantize weight parameters to a relatively small number of shared weights without loss of accuracy. Chen *et al*. [5] introduced a hash function to group network connections into hash buckets and forced connections falling into the same buckets to share the same weight. Other works attempt to reduce the precision of parameter by introducing binary [41, 7, 6] and ternary [55] weights.

**Filter Decomposition:** Apart from the two aforementioned methods, filter decomposition is proposed to approximate the original filter with parameter efficient representations [8, 20, 26, 52, 46, 40]. Early low-rank approximation applies matrix decomposition by using SVD [8] or CP-decomposition [26]. Jaderberg *et al*. [20] proposed to approximate the 2D filter set by a linear combination of a smaller basis set of 2D separable filters. Wang *et al*. [46] built on the work of Jaderberg *et al*. and further rearranged the decomposed filter sequentially. In their work, each normal convolution is decomposed into several layers of depth-wise convolution followed by $1 \times 1$ convolution. Son *et al*. [45] proposed to use k-means algorithm to cluster the $3 \times 3$ convolutional kernels. The kernels that fall in the same cluster share the same weight parameter. However, for each $3 \times 3$ kernel, a scale and an index parameter is introduced to represent the kernel. So the compression ratio in terms of number of parameters is fixed and slightly larger than 2/9. The same problem exists for [46]. Although the compression ratio 1/9 could be achieved by [46], the classification accuracy is severely diminished. Another drawback of [45] is that it could not be applied to $1 \times 1$ convolutions favored by modern networks such as ResNet and DenseNet.

Instead of working on 2D filters as in the previous low-rank approximation, Zhang *et al*. [52] directly dealt with 3D filters by considering the input channel as the third dimension. However, their method cannot reduce the input

channel. This prohibits the application of the decomposition method narrow networks with small output channel but large input channel such as DenseNet. In a recent work, Peng *et al*. [40] proposed to approximate a normal convolution by group convolution followed by a linear combination ($1 \times 1$ convolution). However, they did not apply their approximation methods to DenseNet, which is of particular interest in the newly proposed architectures.

By contrast, our proposed basis learning method can be applied to convolutions with any kernel size and any input/output channel size. This makes our method flexible to compress different modern networks.

## 3. Filter Decomposition for Network Compression

Given an input image $x \in \mathcal{X}$, the aim of supervised learning is to recover the corresponding label $y \in \mathcal{Y}$. For low-level vision tasks such as image SR, the label is the ground-truth high-resolution image corresponding to the low-resolution input image $x$. For high-level image classification, $y$ is a class label of the image. The regression process can be represented by a simple function

$$\hat{y} = f_{\Theta}(x), \tag{1}$$

where $\hat{y}$ denotes the regressed label and $f_{\Theta}(\cdot)$ is the regression function of the neural network parameterized by $\Theta$.

### 3.1. Decomposing convolution layer with filter basis

We assume that a convolution layer has $c$ input channels and $n$ output channels, and the kernel size is $w \times h$. In order to reduce the number of parameters in neural network, different decomposition methods have been suggested. Zhang *et al*. assumed the parameters of a convolution layer could be approximated by a low-rank matrix [52], *i.e*.,

$$\mathbf{W} \approx \mathbf{B} \cdot \mathbf{A}, \tag{2}$$

where $\mathbf{W} \in \Re^{cwh \times n} = [\mathbf{W}_1, \cdots, \mathbf{W}_n]$ is the matrix that contains the vectorized 3D filters, the multiplication of matrix $\mathbf{B} \in \Re^{cwh \times m}$ and matrix $\mathbf{A} \in \Re^{m \times n}$ is a low-rank matrix with rank $m < n$. Besides formulating the parameters of convolution layer as a $cwh \times n$ matrix, there are also other low-rank approximation works [20, 46] which consider the parameter matrix as a $wh \times cn$ matrix. These works treat each channel in the 3D filter independently.

In Eqn. (2), the approximation of filter can also be analyzed in a filter basis decompostion perspective. Each 3D filter $\mathbf{W}_i \in \Re^{cwh \times 1}$ (or $\mathbf{W}_i \in \Re^{wh \times 1}$ for the channel-wise decomposition case) is represented by the linear combination of a set of $m$ filter basis $\{\mathbf{B}_j | j = 1, \cdots, m\}$ with the coding coefficient vector $\mathbf{A}_i \in \Re^{m \times 1}$:

$$\mathbf{W}_i \approx \sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j, i = 1, \cdots, n. \tag{3}$$

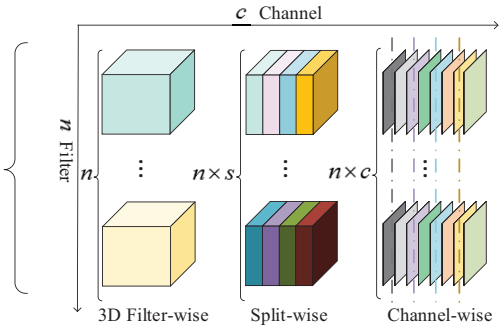


Figure 1. Comparison of different filter decomposition methods. *Right:* each channel of the 3D filter is considered as a basic element. A unique set of basis is learned for the $n$ 2D filters in each channel. *Middle (the proposed):* the 3D filters are split into $s$ groups along the channel dimension and each group is considered as a basic filter element. A basis set is learned for all of the $n \times s$ splits of all the 3D filters. *Left:* the 3D filter is considered as a whole. A basis set is learned for the 3D filters.

where $\mathbf{A}_i$ is the $i$-th column of $\mathbf{A}$, $\mathbf{B}_j$ is the $j$-th filter basis with dimension $cwh \times 1$ or $wh \times 1$ for the 3D filter-wise decomposition and 2D channel-wise decomposition cases, respectively. An illustration of direct 3D filter-wise decomposition and channel-wise filter decomposition can be found in the left and right part of Fig. 1.

From the viewpoint of filter basis decomposition, more flexible decomposition strategy can be adopted. In the next subsection, we analyze the relationship between the dimension of filter basis and compression rate, and suggest a split-wise decomposition approach for network compression.

### 3.2. Compression rate with different filter basis

If we utilize $m$ 3D filter basis as a basic element (Fig 1: *Left*) to decompose the parameters of a convolution layer, the compression rate of the parameters is

$$\frac{m \cdot c \cdot w \cdot h + m \cdot n}{n \cdot c \cdot w \cdot h} = \frac{m}{n} + \frac{m}{c \cdot w \cdot h}, \tag{4}$$

where $(n \cdot c \cdot w \cdot h)$, $(m \cdot c \cdot w \cdot h)$, and $(m \cdot n)$ is the number of parameters of the original convolution layer, the filter basis, and the coding coefficients, respectively. In most of existing neural networks, $c \cdot w \cdot h$ is much larger than $n$. Thus, the first term in Eqn. (4) dominates the compression rate. For the 2D channel-wise decomposition case, we can similarly get the compression rate, namely,

$$\frac{m \cdot w \cdot h + c \cdot m \cdot n}{n \cdot c \cdot w \cdot h} = \frac{m}{n \cdot c} + \frac{m}{w \cdot h}. \tag{5}$$

The major storage budget is used for the coding coefficients.

In order to achieve a better trade-off between compressing the basis and coefficients, we split the 3D filters along the channel dimension as illustrated in the middle part of

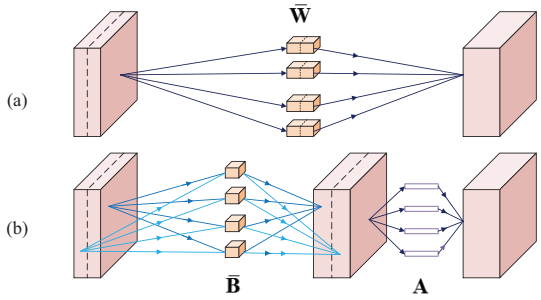


Figure 2. Illustration of the proposed basis learning method. Operations are converted to convolutions. Unlike the normal convolution, our method splits both the input feature map and the 3D filter along the channel dimension. A set of basis is learned for the ensemble of splits. Every split of the input feature map is convolved with the basis. A final $1 \times 1$ convolution generates the output.

Fig. 1, namely, thinking of the $c \times w \times h$ filter as being composed of $s$ smaller $p \times w \times h$ filters and $c = s \cdot p$. As a result, the $n$ 3D $c \times w \times h$ filters can be regarded as $n \cdot s$ filters with size $p \times w \times h$. Then, the problem becomes learning the basis and the representation coefficients of the $n \cdot s$ smaller filters. And the compression rate becomes

$$\frac{m \cdot p \cdot w \cdot h + m \cdot n \cdot s}{n \cdot c \cdot w \cdot h} = \frac{m}{n \times s} + \frac{m}{p \cdot w \cdot h}. \quad (6)$$

The compression rate equation in Eqn. (6) enable us to utilize generalized split-wise decomposition formula to achieve better compression rate. Concretely, the optimal compression rate with respect to the size of filter basis could be achieved by solving the following optimization problem:

$$\begin{aligned} \{s^*, p^*\} &= \arg\min_{\{s,p\}} \left\{ \frac{m}{n \times s} + \frac{m}{p \cdot w \cdot h} \right\} \quad \text{s.t.} \quad c = s \cdot p \\ &= \left\{ \sqrt{\frac{c \cdot w \cdot h}{n}}, \sqrt{\frac{n \cdot c}{w \cdot h}} \right\}. \end{aligned} \quad (7)$$

We can further quantize $p$ to the nearest integer that can divide $c$. For most of the convolutional layers, the input channel $c$ and output channel $n$ are the same or of the same magnitude order, *i.e.*, $c \approx n$. Thus, the optimal group $s^* \approx \sqrt{w \times h}$. That is to say, the optimal configuration of splits is neither Fig. 1: *Left* nor Fig. 1: *Right* but the middle state between them.

### 3.3. Implementing with convolution

In this subsection, we show that filter decomposition can be implemented by convolution in the forward pass. By rearranging the operation, the proposed filter decomposition approach can alleviate the computation burden and compress network parameters.

We start with the case where there is only one split, *i.e.*, $s = 1$. As in Eqn. 3, we utilize linear combination of filter basis to reconstruct the 3D filter $\mathbf{W}_i = \sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j$.

For the simplicity of notation, we use the same notation to represent the original non-vectorized 3D filters and basis, *i.e.*, $\mathbf{W}_i, \mathbf{B}_j \in \Re^{c \times w \times h}$. Thus, the convolution between the input feature map $x$ and the 3D kernel becomes

$$x * \mathbf{W}_i = x * \left( \sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j \right) = \sum_{j=1}^{m} \alpha_{j,i} \left( x * \mathbf{B}_j \right). \quad (8)$$

The second equality follows the linearity of convolution. Eqn. (8) decompose the convolution operation with 3D filter $\mathbf{W}_i$ as linear combination of convolution operations with filter basis $\{\mathbf{B}_j, j = 1, \ldots, m\}$. The linear combination can be implemented by a $1 \times 1$ convolution.

For more general split-wise decomposition cases, we use smaller filter basis $\{\overline{\mathbf{B}}_j \in \Re^{p \times w \times h}, j = 1, \ldots, m\}$ to reconstruct each sub-part of the 3D filter, namely,

$$\mathbf{W}_i = \left[ \overline{\mathbf{W}}_{i,1}; \ldots; \overline{\mathbf{W}}_{i,s} \right], \quad (9)$$

$$\overline{\mathbf{W}}_{i,g} = \sum_{j=1}^{m} \alpha_{j,i,g} \overline{\mathbf{B}}_j, \quad (10)$$

where $\overline{\mathbf{W}}_{i,g} \in \Re^{p \times w \times h}$ is a split of the 3D filter, $g = 1, \ldots, s$ is the split index, and $[\cdot]$ is the operator that stacks the basis along the channel dimension. Accordingly, the convolution between $x$ and $\overline{\mathbf{W}}_i$ becomes

$$\begin{aligned} x * \mathbf{W}_i &= [\overline{x}_1, \cdots, \overline{x}_s] * \left[ \overline{\mathbf{W}}_{i,1}, \cdots, \overline{\mathbf{W}}_{i,s} \right] \\ &= \sum_{g=1}^{s} \overline{x}_g * \overline{\mathbf{W}}_{i,g} = \sum_{g=1}^{s} \overline{x}_g * \sum_{j=1}^{m} \alpha_{j,i,g} \overline{\mathbf{B}}_j \\ &= \sum_{g=1}^{s} \sum_{j=1}^{m} \alpha_{j,i,g} \left( \overline{x}_g * \overline{\mathbf{B}}_j \right). \end{aligned} \quad (11)$$

where $\{\overline{x}_g, g = 1, \ldots, s\}$ in Eqn. (11) are the splits of the input $x$. As revealed by Eqn. (11), in the split-wise decomposition case, each split of feature map is firstly convolved with the filter basis, and then the final output is achieved by a weighted summation of the convolution results. This operation on feature map splits could be implemented as a 3D convolution as in Pytorch [39] or Tensorflow [1] with stride $p = c/s$ and no padding along the channel dimension. But we find the 3D convolution implementation is not efficient. In this way it takes 121 ms to run the compressed EDSR model for one iteration with batch size 16 and patch size $48 \times 48$. Instead, we implement the operation with $s$ 2D convolutions that share the same weight parameter and the running time drops to 62 ms. The linear combination is again converted to a $1 \times 1$ convolution. Thus, no matter how many splits there are, a standard convolution can be decomposed into a convolution with respect to the basis and a $1 \times 1$ convolution. The implementation is illustrated in Fig. 2.

### 3.4. Filter basis decomposition for special filter sizes

As shown in the above analysis, our basis learning method follows a general setting of filter size, *i.e.*, $n \times c \times w \times h$. This means that the proposed basis learning method can be applied to any convolutional filters. Here we emphasize two special filter sizes.

$1 \times 1$ **convolution:** The first one is $1 \times 1$ convolution which is favored by modern neural networks [15, 18]. When the input/output channels are quite large, considerable parameters and computation are consumed by $1 \times 1$ convolution. For example, in DenseNet-12-40 architecture [18], 12.1% of the parameters is in the two large $1 \times 1$ convolutions. Unfortunately, prior filter decomposition works [20, 8, 46] could not be applied to this kind of convolution. Following our formulation Eqn. (8) through Eqn. (11), a $1 \times 1$ convolution with large $n$ and $c$ can be decomposed into two cheaper ones.

$c \gg n > m$ **convolution:** In some networks such as DenseNet, the output channel $n$ is much smaller than the input channel $c$. In this case, according to Eqn. (4), we are in the dilemma of either choosing a even smaller basis size $m$ at the risk of losing too much accuracy or selecting an $m$ comparable with $n$ thus resulting in uneconomic compression. As revealed by Eqn. (5), by splitting the 3D kernels along the channel dimension, we can have $s$ times more filters. So we can gracefully choose a comfortable basis size that leads to both economic compression and high accuracy.

## 4. Learning Filter Basis

In the previous section, we have shown that decomposing filter splits into linear combinations of filter basis could reduce the computational burden and parameter number of networks. In this section, we present our learning method for learning filter basis.

### 4.1. General filter basis learning approach

For the purpose of notation simplicity, we only introduce the simple case of using $\mathbf{B} \cdot \mathbf{A}$ to approximate filter $\mathbf{W}$. The training method for the split-wise case of approximating $\overline{\mathbf{W}}$ with $\overline{\mathbf{B}} \cdot \overline{\mathbf{A}}$ is exactly the same. We jointly minimize the approximation error $\|\mathbf{W} - \mathbf{B} \cdot \mathbf{A}\|_F^2$ and the network target loss $\mathcal{L}(y, f(x))$. For example, to compress image restoration network with mean square error (MSE) loss, our training objective function is

$$\min_{\mathbf{B}^l, \mathbf{A}^l} \left\| y - f_{\mathbf{B},\mathbf{A}|\Theta}(x) \right\|_F^2 + \gamma \sum_{l=1}^{L} \left\| \mathbf{W}^l - \mathbf{B}^l \cdot \mathbf{A}^l \right\|_F^2, \tag{12}$$

where $f_{\mathbf{B},\mathbf{A}|\Theta}(\cdot)$ denotes the CNN with parameter $\{\mathbf{B}, \mathbf{A}\}$ conditioned that the other parameters $\Theta$ are known and the superscript $l$ indexes the $l$-th layer of an $L$-layer network.

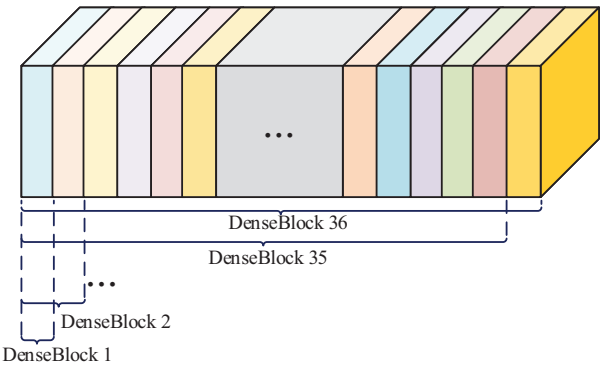


Figure 3. Basis sharing for the compression of DenseNet-12-40. The basis set is shared by all of the DenseBlocks in DenseNet-12-40. The shared basis is split into 36 splits. The basis of a certain DenseBlock is sliced from the shared basis set. Starting from the lower DenseBlock, every DenseBlock adds a new split from the shared basis set to the basis of the previous block forming the basis of current block. Thus, the basis channel of the DenseBlock grows gradually.

After having learned the basis and the coding matrices $\{\mathbf{B}, \mathbf{A}\}$, there is no need to store the original filters. During inference, $\{\mathbf{B}, \mathbf{A}\}$ is used as the weight parameter as the lightweight and $1 \times 1$ convolution, respectively. The total number of parameters of the basis and the coding coefficient is much fewer than those of the original filters, thus achieving a reduction of the number of parameters.

### 4.2. Basis sharing

To compress the networks further, we can force several or all convolutional layers to share the same basis set depending on the compression degree we want to achieve. The weight sharing strategy can be customized to the networks. For example, in ResNet [15] and the following works SRResNet [27], EDSR [33], there are two convolutions in the residual block. We can let the two convolutions share the basis. In ResNet-56 architecture for CIFAR10, the residual blocks are grouped into three groups, each with 9 residual blocks and increasing feature map channels. The channels in the lower residual block groups are relatively small (16 and 32 for the first and second group). To achieve a satisfying compression rate, we have the convolutions within the same group share a common basis set. Moreover, in DenseNet, the input channel grows gradually with a step 12. There is no clear sign like in ResNet to indicate which convolution should share the basis. In this case, all of the convolutional layers share the same basis. For the lower layers, only a slice of the basis is used while only for the last convolutional layer the whole basis is used. The basis sharing strategy for DenseNet-12-40 is shown in Fig. 3.

In conclusion, we can apply block-wise, group-wise, or network-wise basis sharing flexibly according to the architecture of the target network.

Ground Truth: PSNR (dB) Factor [46]: 34.67 Basis-S (ours): 34.99 Basis (ours): 35.10 Baseline: 35.27

Figure 4. SR results of *bird* image for upscaling factor $\times 4$. Network compression methods are applied on EDSR [33].

| Metrics | Basis Share | | | Baseline |
|---|---|---|---|---|
| | No / Yes $m = 16$ | No /Yes $m = 32$ | No / Yes $m = 64$ | |
| Set5 | 32.14 / 32.16 | 32.22 / 32.20 | 32.33 / 32.30 | 32.48 |
| Set14 | 28.58 / 28.57 | 28.66 / 28.64 | 28.72 / 28.73 | 28.81 |
| B100 | 27.58 / 27.57 | 27.62 / 27.61 | 27.66 / 27.64 | 27.72 |
| Urban100 | 26.05 / 26.00 | 26.20 / 26.20 | 26.38 / 26.38 | 26.65 |
| DIV2K | 28.96 / 28.93 | 29.06 / 29.04 | 29.14 / 29.14 | 29.25 |
| #Params | 27k / 17k | 53k / 35k | 106k / 70k | 1180k |
| Comp. (%) | 2.3 / 1.5 | 4.5 / 3.0 | 9.0 / 5.9 | 100 |

Table 1. Different operating points of applying the proposed basis learning to EDSR for image SR for upscaling factor $\times 4$. PSNR (dB) is reported for the five commonly used datasets. 'Basis Share' indicates whether the two convolutions within the same residual block share the basis. $m$ is the number of basis. The number of splits $p$ for one convolution is 4.

## 5. Experimental Results

We show the experimental results in this section and compare with the state-of-the-art methods on both image classification and image SR. For classification, we applied our basis learning method to various networks including VGG [44], ResNet [15], and DenseNet [18]. We evaluate the performance of compressed models on CIFAR10 [24] dataset. The training and testing subset contains 50,000 and 10,000 images, respectively. As is done by prior works [15, 18], we normalize all images using channel-wise mean and standard deviation of the the training set. Standard data augmentation is also applied. We train the compressed networks for 300 epochs with SGD optimizer and an initial learning rate of 0.1. The learning rate is decayed by 10 after 50% and 75% of the epochs.

For image SR, we applied our method to two typical SR networks, namely, SRResNet [27] and EDSR [33]. SRResNet is a middle-level network with 1.5M parameters while EDSR is quite a huge network with 43M parameters but much higher PSNR accuracy. For fast training, we also compressed a lighter version of EDSR with 8 residual blocks and 128 channels per convolution in the residual block. We denote this network as EDSR-8-128. The networks are trained on DIV2K [2] dataset that contains 1,000 2K images. We test the networks on five datasets: Set5 [4], Set14 [48], B100 [36], Urban100 [19], and DIV2K validation set. Adam optimizer [23] is used for training SR networks. We use the default hyper-parameter. The networks are trained for 300 epochs. The learning rate starts from $1 \times 10^{-4}$ and decays by 10 after 200 epochs.

### 5.1. Validation on super-resolution

The compression results of SR networks are shown in Table 1, Table 2, and Table 3. In Table 1, we explore different operating points applied to EDSR. We use 4 splits for the convolution in the residual block. For a clearer comparison, we report the number of parameters and compression ratio for one residual block since all of the other blocks has the same parameter. And we keep to this setting in Table 2 and Table 3. By default, each convolution layer uses an unique basis set, *i.e.*, without basis sharing. In Table 1 and Table 3, the corresponding basis sharing result is also shown.

In Table 1, there are several noticeable points. Firstly, the compressed models with and without basis sharing technique achieve almost the same PSNR for different $m$. But with basis sharing, the model size is further compressed. Secondly, when $m = 64$ and basis sharing is used, the compressed model only accounts for 9% of the parameters of the original network. When basis sharing is further used, an impressive compression rate of 5.9% is achieved while the PSNR result is not far away from the baseline. Thirdly, the most aggressive compression ratio is 1.5%. Considering that there are 32 and 16 residual blocks in EDSR and SRResNet respectively, this operating point brings the model size from EDSR level to SRResNet level while the PSNR of the resulting model is higher than that of SRResNet.

In Table 2, the results of Factor [46] and our basis learning method are shown for the SRResNet and EDSR-8-128. A lighter and a heavier operating point are reported for each compression method. The proposed method outperforms Factor under the two settings. To further compare Factor and the proposed method, we apply the compression method to the fully-fledged EDSR model. As shown in Table 3, the compressed model Basis-S is much better than Factor and in the meanwhile with fewer parameters. The PSNR result of our Basis-S is slightly worse than that of Ba-

| SRResNet [27] | | Factor-SIC2 | Factor-SIC3 | Basis-64-14 (ours) | Basis-32-32 (ours) | Baseline |
|---|---|---|---|---|---|---|
| PSNR (dB) | Set5 | 31.68 | 31.86 | 31.84 | 31.90 | 32.03 |
| | Set14 | 28.32 | 28.38 | 28.38 | 28.43 | 28.50 |
| | B100 | 27.37 | 27.40 | 27.39 | 27.44 | 27.52 |
| | Urban100 | 25.47 | 25.58 | 25.54 | 25.65 | 25.88 |
| | DIV2K | 28.59 | 28.65 | 28.63 | 28.69 | 28.85 |
| #Parameters | | 19k | 28k | 18k | 27k | 74k |
| Compression Rate (%) | | 25.3 | 38.0 | 24.3 | 36.1 | 100 |
| EDSR-8-128 [33] | | Factor-SIC2 | Factor-SIC3 | Basis-128-27 (ours) | Basis-128-40 (ours) | Baseline |
| PSNR (dB) | Set5 | 31.82 | 31.96 | 31.95 | 32.03 | 32.10 |
| | Set14 | 28.40 | 28.47 | 28.42 | 28.45 | 28.55 |
| | B100 | 27.43 | 27.49 | 27.46 | 27.50 | 27.55 |
| | Urban100 | 25.63 | 25.81 | 25.76 | 25.81 | 26.02 |
| | DIV2K | 28.70 | 28.81 | 28.76 | 28.82 | 28.93 |
| #Parameters | | 70k | 105k | 69k | 102k | 295k |
| Compression Rate (%) | | 23.8 | 35.7 | 23.4 | 34.7 | 100 |

Table 2. Comparison of Factor [46] and our basis learning method for the lighter SR networks SRResNet and EDSR-8-128. The upscaling factor is $\times 4$. For each method, two operating points including a lighter one and a heavier one are reported. 'SIC*' denotes the number of SIC layers in Factor. 'Basis-N-S' means that the number of basis is S and each basis has N input channels.

| Comparison Metrics | | Factor [46] | | | Basis-S (ours) | | | Basis (ours) | | | Baseline | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ |
| PSNR (dB) | Set5 | 37.95 | 34.33 | 32.05 | 38.09 | 34.47 | 32.29 | 38.12 | 34.55 | 32.39 | 38.19 | 34.68 | 32.48 |
| | Set14 | 33.53 | 30.31 | 28.54 | 33.75 | 30.41 | 28.63 | 33.72 | 30.46 | 28.69 | 33.95 | 30.53 | 28.81 |
| | B100 | 32.15 | 29.08 | 27.55 | 32.23 | 29.15 | 27.62 | 32.27 | 29.18 | 27.64 | 32.35 | 29.26 | 27.72 |
| | Urban100 | 31.99 | 28.10 | 25.98 | 32.38 | 28.39 | 26.25 | 32.46 | 28.51 | 26.36 | 32.97 | 28.81 | 26.65 |
| | DIV2K | 34.60 | 30.91 | 28.92 | 34.77 | 31.06 | 29.06 | 34.84 | 31.11 | 29.13 | 35.03 | 31.26 | 29.25 |
| #Parameters | | 136k | | | 90k | | | 164k | | | 1180k | | |
| Compression Ratio (%) | | 11.5 | | | 7.6 | | | 13.9 | | | 100 | | |

Table 3. Compression results for EDSR [33]. Basis-S uses basis sharing for the two convolutions within the same residual block.

| Configuration | Top-1 Error (%) | #Params | Comp.(%) |
|---|---|---|---|
| M24 | 5.69 | 320k | 30.8 |
| M26 | 5.70 | 336k | 32.3 |
| M32 | 5.57 | 383k | 36.8 |
| M36T6 | 5.32 | 331k | 31.8 |
| M38T12 | 5.56 | 326k | 31.3 |
| Baseline | 5.26 | 1041k | 100 |

Table 4. Different operating points of our method for compressing DenseNet-12-40 architecture [18] for image classification. 'M*' and 'T*' means the number of basis in DenseBlock and the splits in the transition block in DenseNet, respectively. The classification error is Top-1 error for CIFAR10.

sis. The super-resolved *bird* images of compressed EDSR by different methods are shown in Fig .4. The image from our compressed model has the highest PSNR and it is very close to the baseline in visual quality as well.

## 5.2. Validation on image classification

In Table 4, we show different operating points for the proposed method applied on DenseNet-12-40 architecture.

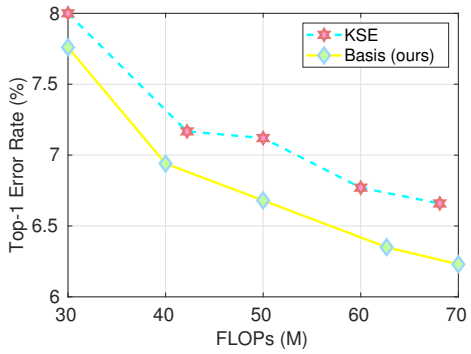


Figure 5. Comparison between our method and KSE [31] for ResNet-56 on CIFAR10.

When the basis size increases from 24 to 32, the corresponding error rate decreases from 5.69% to 5.57%. In addition, by applying compression to the $1 \times 1$ convolution in the transition layer, we can save some parameter budget for the DenseBlock, which is relatively more important in the network. Thus, for 'M36T6' and 'M38T12', we can utilize more basis and at the same time with smaller number of

| Model | Method | Top-1 Error (%) / Baseline | #Parameters | Compression Rate(%) |
|---|---|---|---|---|
| VGG-16 | K-means [45] | 6.24 / 5.98 | 3.27M | 22.2 |
| | Factor [46] | 7.12 / 5.98 | 3.34M | 22.7 |
| | Group [40] | 6.69 / 5.98 | 3.80M | 25.9 |
| | Basis (ours) | 6.18 / 5.98 | 3.21M | 21.8 |
| DenseNet-12-40 | K-means | 5.44 / 5.26 | 335k | 32.2 |
| | Factor | 6.71 /5.26 | 317k | 30.4 |
| | Group | 6.65 / 5.26 | 337k | 32.4 |
| | KSE [31] | 5.30 / 5.19 | 390k | 37.5 |
| | [34](70%) | 5.65 / 5.19 | 350k | 33.6 |
| | Simple-SVD | 7.14 / 5.26 | 360k | 34.6 |
| | Basis (ours) | 5.32 / 5.26 | 331k | 31.8 |
| ResNet-56 | K-means [45] | 6.76 / 6.28 | 190k | 22.4 |
| | Factor | 8.70 / 6.28 | 212k | 24.9 |
| | Group | 6.45 / 6.28 | 206k | 24.3 |
| | KSE | 7.12 / 6.97 | 360k | 42.4 |
| | Basis (ours) | 6.60 / 6.28 | 186k | 21.9 |

Table 5. Compression results for VGG [44], DenseNet [18] and ResNet [15] trained on CIFAR-10 [24]. For a fair comparison, the model size from different methods is kept to the same level.

| Model | Method | Top-1 Err.(%) / Baseline | FLOPs (%) |
|---|---|---|---|
| VGG | K-means [45] | 6.24 / 5.98 | 100 |
| | Factor [46] | 7.12 / 5.98 | 36.6 |
| | Group [40] | 6.69 / 5.98 | 46.1 |
| | Basis (ours) | **6.23** / 5.98 | 23.5 |
| ResNet56 | CaP [37] | 6.78 / 6.49 | 50.2 |
| | ENC [21] | 7.00 / 6.90 | 50.0 |
| | AMC [16] | 8.10 / 7.20 | 50.0 |
| | KSE [31] | 6.77 / 6.97 | 48.0 |
| | Basis (ours) | **6.08** / 7.05 | 50.0 |

Table 6. Top-1 error vs. FLOPs reduction rate for VGG-16 and ResNet-56 on CIFAR10. For K-means, the practical FLOPs in the authors' code rather than the theoretical is reported.

parameters. Compared with 'M32', 'M36T6' further reduces the error rate by 0.25%. Interestingly, although our 'M38T12' model uses two more basis than 'M36T6', the error rate rises a little bit. This is because 'M38T12' uses an aggressive compression, *i.e.*, $s = 12$ in the transition block. Therefore, the compression degree of the DenseBlock and the transition block should be balanced to obtain the best trade-off between compression ratio and accuracy.

The compression results of different methods for VGG-16, DenseNet-12-40, and ResNet-56 are shown in Table 5. For a fair comparison, we follow the setting in [45] for VGG-16. That is, only one instead of three fully-connected layer is appended after the last pooling layer. On VGG-16, our method shows the most aggressive compression and the lowest error rate. For our compressed model, we only suffer 0.2% increase of error rate, which is quite small compared with 0.71% 1.14% increase of Group [40] and Factor. And our model has the smallest size. Our compression method and KSE [31] shoots the lowest error rate on DenseNet-12-40. As for the compression ratio, although Factor is slightly lower than ours, its accuracy is the worst among all the compared methods. For ResNet-56, our method performs comparable with Group in terms of accuracy while with 20k fewer parameters. Table 6 and Fig. 5 compare the computational cost of the proposed method and the state-of-the-art. Results are reported at operating points different from those in Table 5. For VGG-16, our method achieves the lowest error rate under the severest FLOPs reduction. For ResNet-56, the proposed method outperforms the others by a significant margin under the same FLOPs reduction. In Fig. 5, our method always shoots a lower error rate than KSE.

## 6. Conclusion

In this paper, we explored how to learn a filter basis set for convolution operations in modern CNNs. Our method is not limited by the filter size. Thus, it can be applied to $1 \times 1$ convolution and convolution with large input channel and smaller output channel. We applied our basis learning method to image classification and SR networks. The experiments validate the advantage of our basis learning method. Our compressed SRResNet and EDSR outperforms the models from the previous filter decomposition method. For the image SR network EDSR, the most aggressive compression our method brings the model size from EDSR level to SRResNet level while being more accurate than SRResNet. For VGG-16, the error rate of the model compressed by our method is within 0.2% the baseline error, which is much better than the results of other compression methods. Our filter basis learning method leads to state-of-the-art performance on ResNet and DenseNet.

**Acknowledgments:** This work was supported by Huawei, the ETH Zurich General Fund, and an Nvidia GPU hardware grant.

## References


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 4

[2] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *Proc. CVPRW*, July 2017. 6

[3] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Proce. NIPS*, pages 2270–2278, 2016. 2

[4] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proc. BMVC*, 2012. 6

[5] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *Proc. ICML*, pages 2285–2294, 2015. 2

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. NIPS*, pages 3123–3131, 2015. 2

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 1, 2

[8] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proc. NIPS*, pages 1269–1277, 2014. 2, 5

[9] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *Proc. ECCV*, pages 184–199. Springer, 2014. 1

[10] Ross Girshick. Fast R-CNN. In *Proc. ICCV*, pages 1440–1448, 2015. 1

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, pages 580–587, 2014. 1

[12] Shuhang Gu, Radu Timofte, and Luc Van Gool. Multi-bin trainable linear unit for fast image restoration networks. *arXiv preprint arXiv:1807.11389*, 2018. 1

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 2

[14] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Proc. NIPS*, pages 1135–1143, 2015. 1, 2

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016. 1, 5, 6, 8

[16] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *Proc. ECCV*, pages 784–800, 2018. 8

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proc. ICCV*, pages 1389–1397, 2017. 1, 2

[18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. CVPR*, pages 2261–2269, 2017. 1, 5, 6, 7, 8

[19] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proc. CVPR*, pages 5197–5206, 2015. 6

[20] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. BMVC*, 2014. 1, 2, 3, 5

[21] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proc. CVPR*, June 2019. 8

[22] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proc. CVPR*, 2016. 1

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[24] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6, 8

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012. 1

[26] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 2

[27] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. CVPR*, pages 105–114, 2017. 5, 6, 7

[28] Dongwoo Lee, Haesol Park, In Kyu Park, and Kyoung Mu Lee. Joint blind motion deblurring and depth estimation of light field. In *Proc. ECCV*, pages 288–303, 2018. 1

[29] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016. 1

[30] Yawei Li, Eirikur Agustsson, Shuhang Gu, Radu Timofte, and Luc Van Gool. CARN: convolutional anchored regression network for fast and accurate single image super-resolution. In *Proc. ECCVW*, pages 166–181. Springer, 2018. 1

[31] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable CNN compression. In *Proc. CVPR*, 2019. 7, 8

[32] Yawei Li, Vagia Tsiminaki, Radu Timofte, Marc Pollefeys, and Luc Van Gool. 3D appearance super-resolution with deep learning. In *Proc. ICCV*, 2019. 1

[33] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proc. CVPRW*, pages 1132–1140, 2017. 1, 5, 6, 7

[34] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proc. ICCV*, pages 2736–2744, 2017. 8

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, pages 3431–3440, 2015. 1

[36] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, volume 2, pages 416–423, July 2001. 6

[37] Breton Minnehan and Andreas Savakis. Cascaded projection: End-to-end network compression and acceleration. In *Proc. CVPR*, June 2019. 8

[38] Jinshan Pan, Deqing Sun, Hanspeter Pfister, and Ming-Hsuan Yang. Blind image deblurring using dark channel prior. In *Proc. CVPR*, pages 1628–1636, 2016. 1

[39] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in Pytorch. 2017. 4

[40] Bo Peng, Wenming Tan, Zheyang Li, Shun Zhang, Di Xie, and Shiliang Pu. Extreme network compression via filter group approximation. In *Proc. ECCV*, pages 300–316, 2018. 1, 2, 3, 8

[41] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. ECCV*, pages 525–542. Springer, 2016. 1, 2

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proc. CVPR*, pages 779–788, 2016. 1

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. NIPS*, pages 91–99, 2015. 1

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 6, 8

[45] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. Clustering convolutional kernels to compress deep neural networks. In *Proc. ECCV*, pages 216–232, 2018. 1, 2, 8

[46] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *Proc. ICCV*, pages 545–553, 2017. 1, 2, 3, 5, 6, 7, 8

[47] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proc. NIPS*, pages 2074–2082, 2016. 2

[48] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International Conference on Curves and Surfaces*, pages 711–730. Springer, 2010. 6

[49] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: residual learning of deep CNN for image denoising. *IEEE TIP*, 26(7):3142–3155, 2017. 1

[50] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proc. CVPR*, pages 3929–3938, 2017. 1

[51] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Deep plug-and-play super-resolution for arbitrary blur kernels. In *Proc. CVPR*, pages 1671–1681, 2019. 1

[52] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE TPAMI*, 38(10):1943–1955, 2015. 1, 2, 3

[53] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017. 1

[54] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *Proc. ECCV*, pages 662–677. Springer, 2016. 2

[55] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016. 2