Packing Convolutional Neural Networks in the Frequency Domain

Yunhe Wang, Chang Xu, Chao Xu and Dacheng Tao, *Fellow, IEEE*

Abstract—Deep convolutional neural networks (CNNs) are successfully used in a number of applications. However, their storage and computational requirements have largely prevented their widespread use on mobile devices. Here we present a series of approaches for compressing and speeding up CNNs in the frequency domain, which focuses not only on smaller weights but on all the weights and their underlying connections. By treating convolution filters as images, we decompose their representations in the frequency domain as common parts (*i.e.*, cluster centers) shared by other similar filters and their individual private parts (*i.e.*, individual residuals). A large number of low-energy frequency coefficients in both parts can be discarded to produce high compression without significantly compromising accuracy. Furthermore, we explore a data-driven method for removing redundancies in both spatial and frequency domains, which allows us to discard more useless weights by keeping similar accuracies. After obtaining the optimal sparse CNN in the frequency domain, we relax the computational burden of convolution operations in CNNs by linearly combining the convolution responses of discrete cosine transform (DCT) bases. The compression and speed-up ratios of the proposed algorithm are thoroughly analyzed and evaluated on benchmark image datasets to demonstrate its superiority over state-of-the-art methods.

Index Terms—CNN compression, Discrete Cosine Transform, Frequency domain speed-up, DCT bases.

1 Introduction

Thanks to the large amount of accessible training data and computational power of GPUs, deep learning models, especially convolutional neural networks (CNNs), have been successfully applied to various computer vision (CV) applications such as image classification [41], [51], [5], visual recognition [42], [9], [27], image processing [14], and object detection [17], [37], [38]. However, most of the widely used CNNs can only be launched on desktop PCs or even workstations given their demanding storage and computational resource requirements. For example, over 232MB of memory and over 7.24×10^8 multiplications are required to launch AlexNet and VGG-Net per image, preventing them from being used in mobile terminal apps on smartphones or tablet PCs. Nevertheless, CV applications are growing in importance for mobile device use and there is, therefore, an imperative to develop and use CNNs for this purpose.

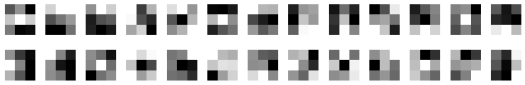
Considering the lack of GPU support and the limited storage and CPU performance of mainstream mobile devices, compressing and accelerating CNNs is essential. Although CNNs can have millions of neurons and weights, a recent research [19] has highlighted that over 85% of weights are useless and can be set to 0 without an obvious deterioration in performance. This suggests that the gap in demands made by heavy CNNs and the limited resources offered by mobile devices may be bridged.

Some effective algorithms have been developed to tackle this challenging problem. [18] utilized vector quantization to allow similar connections to share the same cluster center. [13] showed that the weight matrices can be reduced by low-rank decomposition approaches. [8] proposed a network architecture using the hashing trick and then transferred the HashedNet into the discrete cosine transform (DCT) frequency domain [7] in order to give different codes various weights by exploiting the property of frequency distribution of natural images. [36], [10] proposed binaryNet, whose weights were -1/1 or -1/0/1 [2]. [19] employed pruning [20], quantization, and Huffman coding to obtain a greater than $35\times$ compression ratio and $3\times$ speed improvement, thereby producing state-of-the-art CNNs compression. Although the performance of existing methods have been demonstrated on benchmark datasets and models, some important issues in deep model compression and speed-up remain to be fully addressed:

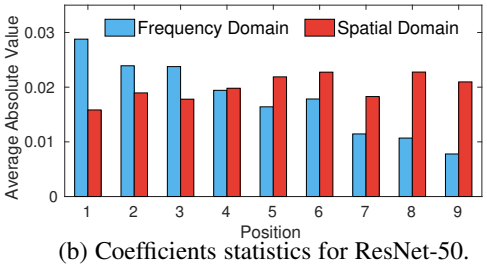
1) **Contextual information.** Independently considering each weight ignores the context information of weights which tends to be helpful for weights compression as well. Although some SVD based methods [13], [28] utilized low rank matrix to approximate original filters, they focused more on only the relationship between filters rather than the internal relationship between weights in a filter. Besides subtle weights, the redundance may also lies in larger weights, *e.g.*, a filter filled with 1 can be represented by only one direct current (DC) frequency coefficient.
2) **Smoothness.** Convolution filters have been suggested to own some intrinsic properties and used to be smooth [55], [34], [7], as shown in Fig. 7. Though convolution filters are often dense-valued in the spatial domain, the energy of their frequency representations used to be concentrated in a few low-frequency components, *i.e.*, lots of high-frequency components are negligibly small. In order to have an explicit illustration, we visualized convolution filters of ResNet-50 in Fig. 1 (a). Although sizes of these filters are very small (3×3), they still have some intrinsic structures. Fig. 1 (b) details average absolute values of original filters and their frequency coefficients,

- *Y. Wang and C. Xu are with the Key Laboratory of Machine Perception (Ministry of Education) and Cooprtative Medianet Innovation Center, School of EECS, Peking University, Beijing 100871, P.R. China. E-mail: wangyunhe@pku.edu.cn, xuchao@cis.pku.edu.cn.*
- *C. Xu and D. Tao are with the UBTech Sydney Artificial Intelligence Centre and the School of Information Technologies in the Faculty of Engineering and Information Technologies at The University of Sydney, J12 Cleveland St, Darlington NSW 2008, Australia. Email: c.xu@sydney.edu.au, dacheng.tao@sydney.edu.au.*

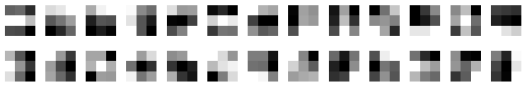
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence



(a) Original convolution filters of ResNet-50.



(b) Coefficients statistics for ResNet-50.



(c) Compressed convolution filters of ResNet-50.

Fig. 1: Visualization of original filters and filters after discarding top-3 high frequency coefficients in the DCT frequency domain.

where 1∼9 indicate frequency positions from low to high. It is clear that values at higher frequency positions are smaller than those at lower frequency positions. In contrast, values in the spatial domain (original filters) are balanced. Moreover, Fig. 1 (c) shows filters after directly discarding top-3 high frequency coefficients. These filters still have similar structures to those in Fig. 1 (a), which suggests the negligible influence of higher frequency coefficients on convolution filters.

3) **Data dependency.** Most of existing approaches accomplish the compression without considering the data input [19], [10], [8]. They assumed that removing smaller weights would have negligible affection on the resulting feature map. However in practice, the value of feature map is determined by the weights of filters as well as the the value of input data. Small weights may be associated with great input values, while large weights can also be connected with subtle input values. The feature map is therefore difficult to remain unchanged by solely considering the filter.

To address these aforementioned problems, we propose to handle convolution filters in the frequency domain using DCT (see Fig. 2) in order to address these aforementioned problems. In practice, convolution filters are designed for extracting intrinsic structures of natural images, and can be usually regarded as small and smooth image patches. Recall that any operation on frequency coefficients of a convolution filter in the DCT domain is equivalent to an operation performed simultaneously over all weights of this filter in the spatial domain. We are thus motivated to explore the contextual property of convolution filters using their frequency coefficients. In order to achieve a higher compression ratio, we factorize the representation of the convolution filter in the frequency domain as the composition of common parts shared with other similar filters and its private part describing some unique information. Both parts can be significantly compressed by discarding a large number of subtle frequency coefficients. Furthermore, we suggest that the compression in the frequency domain can receive valuable guidance from the input data in the spatial domain. Hence, both redundancy in spatial and frequency domains will be explored simultaneously, and more optimal weights compression can be expected.

Meanwhile, we develop an extremely fast convolution calculation scheme that exploits the relationship between the feature maps of DCT bases and frequency coefficients of convolution filters. Specifically, a convolution between the input data and a convolution filter can be realized by a weighted combination of the convolution responses of DCT bases on the input data. Furthermore, we have theoretically discussed the compression and the speed-up of the proposed algorithm. Experimental results on benchmark datasets demonstrate that our proposed algorithm can consistently outperform state-of-the-art competitors, with higher compression ratios and speed gains.

A preliminary version of this work was presented earlier [50], namely CNNpack. The initial version excavates redundant weights in the DCT frequency domain, and the present work adds to the initial version in significant ways. First, we extend the CNNpack to a data-driven method which not only discards subtle weights in convolution filters but also excavates useless weights with tiny responses to real word datasets. Second, we improve the CNNpack by investigating redundancy in both frequency and spatial domains, which provides a greater compression possibility and explore a detailed optimization algorithm for compression deep CNNs. Third, considerable new analyses and intuitive explanations are added to the initial results. In addition, some recently published methods have included for comparing with the proposed method. Experimentally, we demonstrate that the compression performance can be improved in comparison to the initial version and the state-of-the-art methods.

This paper is organized as follows. Section 2 investigates related works on compressing and speeding up CNNs. Section 3 proposes the CNNpack for compressing filters in the DCT frequency domain, and the data-driven method, which investigates the redundancy in spatial and frequency domains simultaneously, is illustrated in Section 4. Section 5 explores a convolution speed-up scheme. Section 6 presents the experimental setup and results of our experimental validation, and Section 7 gives the conclusion of this paper.

2 Related Work

It is well known that, existing deep convolutional neural networks are over-parametrized and there is significant redundancy in their parameters [19], [18], [7], [13], [26]. A variety of related works have been proposed to reduce the storage and complexity of CNNs. Based on techniques they used, compression methods can be divided into three categories.

2.1 Weight Matrix Decomposition

Fully connected layers, which are often placed in the last several layers of the network, *e.g.*, the last three layers of VGGNet-16 [41], used to occupy a large proportion of the storage for the entire CNNs. The output of a fully connected layer can be formulated as $Wx + b$, where x is the input data, each row of matrix W corresponds to the weights of a neuron, and b is the bias. Given the enormous number of neurons (*e.g.*, 4096) and their inner similarity, considerable redundant information often exists in the huge matrix. [13] used singular value decomposition (SVD) technique to discover the low-rank approximation of W [6]. Similarly, [28] regarded filters in a layer as a Tucker and utilized tensor decomposition techniques to speed up CNNs. However, filters are designed for extracting diverse information,

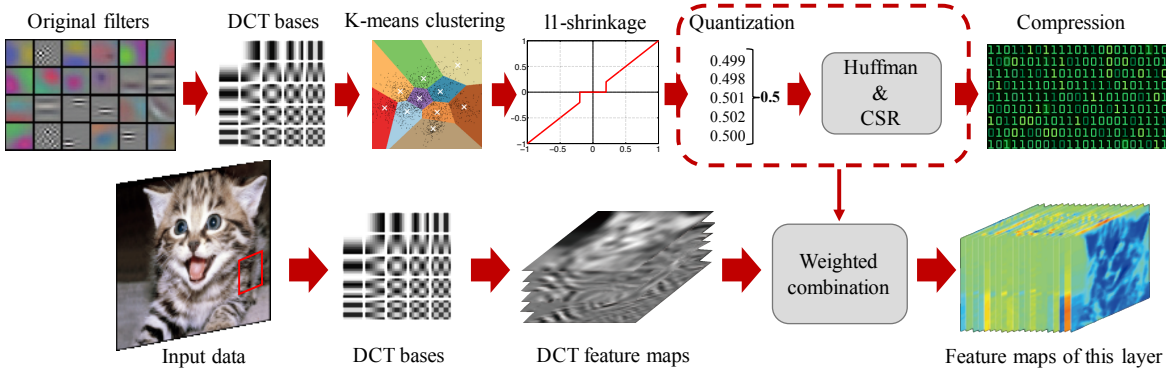


Fig. 2: The diagram of the proposed CNN compression method in the frequency domain. Our method has two pipelines, the top row shows our compression algorithm and the bottom row illustrates the proposed speedup scheme in the DCT frequency domain. It is remarkable to note that the network compressed by the proposed method can be directly used in the frequency domain without decompression.

the intrinsic rank of W might be large in practice. Employing a rather low-rank matrix to represent W thus tends to significantly decline the accuracy of the network. In addition, [30] replaced original convolution filters by rank-1 matrices, and the original square convolutions operations are divided into a series of vector convolutions. Although the performance of the network consisting of rank-1 matrices is inferior to that of the original one, it is encouraged [44] to use vector filters (*i.e.*, 1×7 and 7×1) to construct a deeper network with higher accuracy and similar amount of weights.

Another effective technique for matrix decomposition is to construct a small dictionary using a set of pre-learned bases [26], and then convolution filters can be approximated as a weighted linear combination of basis filters. If the number of basis is much less than the dimensionality of convolution filters, the storage complexity will be reduced significantly. However, since there are lots of filters whose sizes are relatively small, *e.g.*, 3×3 filters in VGG-16 Net [41] and AlexNet [29] or even 1×1 filters in ResNet [23], it is difficult to discover such a small yet accurate dictionary. [3] further proposed to decompose original convolution filters as weighted combinations of basis filters and sparse coefficients thus obtained higher compression and speed-up ratios.

2.2 Weight Quantization

Weight quantization is another important way to remove redundant information in weights of neurons within a well-trained CNN. Its motivation is to excavate the similar information between different areas (*e.g.*, any 1×3 vectors in several 3×3 filters) of convolution filters and represent them using the quantized data[12]. [18] employed k-means to obtain the cluster centers of weights of convolution filters, and then approximately represented convolution filters using their corresponding clustering centers. [8] 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. [7] further proposed weighting the hash codes by DCT coefficients in the frequency domain. These weight sharing schemes indeed provide a considerable compression ratio but they have no contribution to the speedup ratio.

To reduce the cost of 32-bit floating values storage and multiplications, [46] proposed a fixed-point implementation with 8-bit integer values, and [25] explored an optimized fixed-point strategy with ternary weights and 3-bit values. Network binarization is an extreme approach for quantizing weights [10], [11], [49]. It simply sets weights as +1/-1 according to their signs so that 32-bit floating values become 2-bit binary and multiplication between floating values is reduced to that of binary values, which significantly reduces the storage and the computation simultaneously. However, this simple strategy will sacrifice too much accuracy of the network. Afterwards, [2] studied a random-like sparse network with +1/0/-1 weights, which alleviates the hard +1/-1 constraint. [36] proposed a more robust binarization method and a novel XNOR calculation for the binarized net. But these attempts are rather simple and crude which often leads to a significant accuracy drop of the original CNN.

2.3 Weight Pruning

Pruning is a simple yet effective scheme for compressing CNNs [21], [22], [20]. Considering the convolution as a weighted combination of input data and filter weights, the weight whose absolute value is extremely small tends to have limited influence on the resulting feature map. [19] filtered weights in pre-trained CNNs using a given threshold, and the generated sparse networks can be compactly stored in sparse row format (CSR) [4] and Huffman encoding [24]. The computation speedup can be realized, given the sparseness of the stored convolution filters. The effectiveness of the pruning strategy relies on the assumption that if the absolute value of a weight in a CNN is sufficiently small, its influence on the output is often negligible. However, if more than 90% components of convolution filters are discarded, there is a strong probability that some pixels of the input image or its feature maps are ignored. For example in Fig. 7(b), the first components of all filters of a convolutional layer has been discarded, and these filters will not scan and extract information from the first pixel of any local patches.

On the other side, in order to achieve a considerable sparsity of compressed CNNs, [33] learned a set of kernel bases (*i.e.*, a dictionary of convolution filters), and then transferred original filters in the coefficient domain with high sparsity. Thus, the calculation complexity of the sparse network is much lower than that of the original one. In addition, [16] proposed using a mask to make the input data sparse to reduce the computational complexity. [52] excavated redundancy by pruning weights in different aspects (*e.g.*, channels, filters, neurons) resulting in a sparse and compact

CNN for speeding up. Therefore, we are also motivated to explore a more sparse architecture of CNN in the DCT frequency domain.

3 Compressing CNNs in the DCT Frequency Domain

Recently developed CNNs contain a large number of convolution filters. convolution filters often have some intrinsic patterns, so that they can be applied for extracting informative features of input images. We are thus motivated to regard convolution filters as small images with intrinsic patterns, and present an approach to compress CNNs in the frequency domain with the help of the DCT.

3.1 The Discrete Cosine Transform (DCT)

Here, we briefly introduce some backgrounds of the two dimensional DCT for digital images. Different from FFT coefficients, coefficients of DCT are real numbers. DCT plays an important role in JPEG compression [48], which is regarded as an approximate KL-transformation for 2D images [1]. In JPEGs, the original image is usually divided into several square patches, which are then processed in the DCT frequency domain. Given an image patch $p \in \mathbb{R}^{n\times n}$, its DCT coefficient $\mathcal{C} \in \mathbb{R}^{n\times n}$ is defined as:

$$\begin{aligned}\mathcal{C}_{j_1 j_2} &= \mathfrak{D}(p_{i_1 i_2}) \\ &= s_{j_1} s_{j_2} \sum_{i_1=0}^{n-1}\sum_{i_2=0}^{n-1} \alpha(i_1,i_2,j_1,j_2) p_{i_1 i_2} = c_{j_1}^T P c_{j_2},\end{aligned} \tag{1}$$

where $s_j = \sqrt{1/n}$ if $j = 0$ and $s_j = \sqrt{2/n}$, otherwise, and $\mathcal{C} = C^T p C$ is the matrix form of the DCT, where $C = [c_1, ..., c_d] \in \mathbb{R}^{d\times d}$ is the transformation matrix. The basis of this DCT is $S_{j_1 j_2} = c_{j_1} c_{j_2}^T$, where

$$c_j(i) = \left(\frac{\pi(2i+1)j}{2n}\right), \tag{2}$$

and $\alpha(i_1, i_2, j_1, j_2)$ denotes the cosine basis function:

$$\begin{aligned}&\alpha(i_1,i_2,j_1,j_2) = \\ &\quad \cos\left(\frac{\pi(2i_1+1)j_1}{2n}\right)\cos\left(\frac{\pi(2i_2+1)j_2}{2n}\right).\end{aligned} \tag{3}$$

Additionally, the DCT is a linear lossless transformation, which enables us to recover the original image by simply utilizing the inverse DCT, *i.e.*,

$$\begin{aligned}p_{i_1 i_2} &= \mathfrak{D}^{-1}(\mathcal{C}_{j_1 j_2}) \\ &= \sum_{j_1=0}^{n-1}\sum_{j_2=0}^{n-1} s_{j_1} s_{j_2} \alpha(i_1,i_2,j_1,j_2)\mathcal{C}_{j_1 j_2},\end{aligned} \tag{4}$$

whose matrix form is $p = C\mathcal{C}C^T$. Furthermore, to facilitate the notations we denote the DCT and the inverse DCT for vectors as

$$\begin{aligned}vec(\mathcal{C}) &= \mathfrak{D}(vec(p)) = (C\otimes C)vec(p) = \mathbf{S}vec(p), \\ vec(p) &= \mathfrak{D}^{-1}(vec(\mathcal{C})) = \mathbf{S}^T vec(\mathcal{C}),\end{aligned} \tag{5}$$

where $vec(\cdot)$ is the vectorization operation and $\otimes$ is the Kronecker product, $\mathbf{S}$ is the DCT transform which stacks $d \times d$ DCT bases, and $\mathbf{S}$ is an orthogonal matrix, *i.e.*, $\mathbf{S}^T\mathbf{S} = \mathbf{I}$.

3.2 Convolutional Layer Compression

Given the success of DCT in image compression, we are motivated to study the convolutional neural networks compression problem in the DCT frequency domain. Considering the redundancy between filters in a relatively large CNN, filters can be decomposed into common parts and private parts for compact storage. At first, we consider compressing different layers separately. The compression scheme is then extended into a global compression for different layers.

For a given convolutional layer $\mathcal{L}$, we first extract its convolution filters $F = \{F_1, ..., F_N\}$, where the size of each convolution filter is $d \times d$ and N is the number of filters in $\mathcal{L}$. Each filter can then be transformed into a vector, and together they form a matrix $\mathbf{F} = [vec(F_1), ..., vec(F_N)] \in \mathbb{R}^{d^2\times N}$ (here we drop the script of channels for having an explicit presentation).

DCT has been widely used for image compression, since DCT coefficients present an experienced distribution in the frequency domain. Energies of high-frequency coefficients are usually much smaller than those of low-frequency coefficients for 2D natural images, *i.e.*, the high frequencies tend to have values equal or close to zero [48]. Additionally, the main profit in the frequency domain is that we can simultaneously handle all pixels in original filters by only processing one component in the frequency domain. Hence, we propose to transfer $\mathbf{F}$ into the DCT frequency domain and obtain its frequency representation $\mathcal{C} = \mathbf{SF} = [\mathcal{C}_1, ..., \mathcal{C}_N]$, where the i-th column $\mathcal{C}_i$ denotes the frequency representation of the i-th in the DCT domain. The shrinkage in the frequency domain can be easily formulated as

$$\arg\min_{\widehat{\mathbf{F}}} ||\mathbf{F} - \widehat{\mathbf{F}}||_F^2 + \lambda||\mathbf{S}\widehat{\mathbf{F}}||_1, \tag{6}$$

where $\widehat{\mathbf{F}}$ is the desired sparse filter matrix in the DCT frequency domain, $||\cdot||_F$ is the Frobenius norm, $||\cdot||_1$ is the ℓ_1 norm, and λ is a parameter for balancing the reconstruction error and sparsity penalty.

A larger λ makes $\widehat{\mathbf{F}}$ sparser, but reduces more performance of the original network. In order to maintain the performance given a relatively large λ, we propose clustering filters in $\mathbf{F}$ and using cluster centers to retain some of their information. Thus, we first exploit the conventional k-means algorithm on all filters in the frequency domain, *i.e.*, $\mathbf{SF}$, to learn a codebook $\mu = [\mu_1, ..., \mu_K]$ of K cluster centers. The memory usage of the original network will be significantly reduced by representing similar filters using the corresponding cluster center, however the accuracy of the original network will significantly decrease as well [18]. Besides the clustering centers, we have to restore the residuals to retain the accuracy. For each convolution filter, we divide its frequency representation into common (cluster centers) and private parts (residuals) as:

$$\mathbf{SF}_j = \mathcal{R}_j + \mu_{k_j}, \quad \mathbf{SF} = \mathcal{R} + U, \tag{7}$$

where $k_j = \arg\min_k ||\mathbf{SF}_j - \mu_k||_2$ is the index of the closest cluster center, $\mathcal{R} = [\mathcal{R}_1, ..., \mathcal{R}_{N_i}]$ stacks residual data in the i-th convolutional layer, and $U = [\mu_{k_1}, ..., \mu_{k_N}]$ are corresponding cluster centers. Then, we first fix cluster centers and employ the ℓ_1 shrinkage to the residuals and reformulate Eq. 6 as:

$$\begin{aligned}&\arg\min_{\widehat{\mathcal{R}}} ||\mathbf{F} - \widehat{\mathbf{F}}||_F^2 + \lambda||\widehat{\mathcal{R}}||_1 \\ &= \arg\min_{\widehat{\mathcal{R}}} ||\mathbf{S}^T\mathcal{R} - \mathbf{S}^T\widehat{\mathcal{R}}||_F^2 + \lambda||\widehat{\mathcal{R}}||_1 \\ &= \arg\min_{\widehat{\mathcal{R}}} ||\mathcal{R} - \widehat{\mathcal{R}}||_F^2 + \lambda||\widehat{\mathcal{R}}||_1,\end{aligned} \tag{8}$$

where $\widehat{\mathcal{R}}$ is the desired sparse residual in the frequency domain, which is given by:

$$\widehat{\mathcal{R}} = \text{sign}(\mathcal{R}) \odot \max\{|\mathcal{R}| - \frac{\lambda}{2}, 0\}, \tag{9}$$

where $\text{sign}(\cdot)$ is the sign function and $\odot$ is the element wise product. In addition, cluster centers also account for a considerable proportion of the amount of the data thus we also need to compress cluster centers for having higher compression ratios. Therefore, we employ the ℓ_1-shrinkage to the cluster centers before Eq. 7 to remove redundancy, *i.e.*,

$$\widehat{\mu} = \text{sign}(\mu) \odot \max\{|\mu| - \frac{\lambda}{2}, 0\}, \tag{10}$$

and reformulate $\widehat{U} = [\widehat{\mu}_{k_1}, ..., \widehat{\mu}_{k_N}]$ according to indexes of filters.

The sparse data obtained through Eq. 9 and Eq. 10 are continuous, which is not benefit for storing and compressing. Hence we need to represent similar values with a common value, *e.g.*, $\{0.101, 0.100, 0.102, 0.099\} \rightarrow 0.100$. Inspired by the conventional JPEG algorithm, rounding coefficients after dividing a large integer will significantly discard subtle and useless information. We use the following function to quantize the residual data:

$$\overline{\mathcal{R}} = \mathcal{Q}\left(\widehat{\mathcal{R}}, \Omega, b\right) = \frac{\mathcal{I}\left\{\Omega \cdot \text{Clip}(\widehat{\mathcal{R}}, -b, b)\right\}}{\Omega}, \tag{11}$$

where $\text{Clip}(x, -b, b) = \max(-b, \min(b, x))$ with boundary $b > 0$, and Ω is a large integer with similar functionality to the quantization table in the JPEG algorithm.

It is obvious that the quantized values in the given network resulting from Eq. 11 are repeated thus we can construct a codebook using all the unique values in them. Since occurrence probabilities of elements in the codebook are unbalanced, representing and storing them in the same code length is inefficient. Huffman encoding is therefore introduced for a more compact storage. Moreover, there are numerous zeros in the data after Huffman encoding, *i.e.*, it is an extremely sparse matrix. Hence the Huffman encoded data is stored in the compressed sparse row format (CSR), denoted as $\mathbf{E}_i$. In addition, cluster centers $\widehat{\mu}$ will be quantized and encoded to $\overline{\mu}$ in the same manner to reduce the storage requirement. Note that the same Huffman dictionary have been used for the sparse cluster centers and residual data, since they empirically follow the same distribution, since filters are initialized by random Gaussian numbers [47] and residual data generally follows a Gaussian distribution with zero expectation due to the squared objective function in k-means.

Generally, the performance of the compressed CNN is inferior to that of the original one, but it has been shown that a fine-tuning operation after compression can enhance the accuracy of the compressed network [20], [19] to a certain degree. In the proposed algorithm, we also employ the fine-tuning approach by fixing weights that have been discarded, *i.e.*, weights pruned by Eq. 10 will not change. Thus, the fine-tuning operation does not decrease the compression ratio. After generating a new model in each iteration, we apply Eq. 11 again to quantize its parameters until convergence.

The above scheme for compressing convolutional neural network has to store data matrices, including the compressed residual data $\mathbf{E}$ and Huffman dictionary with H quantized values, the compressed $\tilde{\mathbf{E}}$ composed of the k-means centers. Given a network with p convolution layers $\{\mathcal{L}_1, ..., \mathcal{L}_p\}$, the number of filters in the i-th layer is N_i with the filter size of $d_i \times d_i$. Weights in CNN used to be stored in 32-bit floating-point, thus the amount of the data for storing the original convolutional layer $\mathcal{L}_i$ is $32N_id_i^2$. As for the network after applying the proposed scheme, we store the Huffman dictionary in 32-bit floating-point to maintain the precision of the network, and we only need $\log K$ bits to encode indexes of cluster centers. The compression ratio of the proposed approach for the given CNN can thus be calculated as:

$$r_c = \frac{\sum_{i=1}^{p} 32N_id_i^2}{\sum_{i=1}^{p}\left(N_i \log K + B_i + \tilde{B}_i + 32H_i\right)}, \tag{12}$$

where B_i and $\tilde{B}_i$ are bits to store $\mathbf{E}_i$ and cluster centers, respectively. H_i stands for the bits to store the Huffman dictionary (*i.e.*, one-dimensional cluster centers).

Since sizes of filters in different convolutional layers are various, we need to apply p times k-means algorithm and learn p Huffman dictionaries with p hyper-parameters (number of clusters) accordingly. Given the fact that filters are initialized by Gaussian random numbers [47], the range of filters in different layers of a well developed network tends to be consistent. If filters from different layers share the same cluster centers and Huffman dictionary, the compression ratio of the proposed approach will be reduced significantly. Especially, there are more than 30 convolutional layers in current CNNs [43], [23]. To enable all convolution filters to share the same cluster centers $U \in \mathbb{R}^{d_i^2 \times k}$ in the frequency domain, we must convert them into a fixed-dimensional space. It is intuitive to directly resize all convolution filters into matrices of the same dimensions and then apply k-means. However, in the spatial-domain, this simple resizing method is not applicable. Considering $\overline{d}$ as the target dimension and $d_i \times d_i$ as the convolution filter size of the i-th layer, the weight matrix would be inaccurately reshaped in the case of $d_i < \overline{d}$ or $d_i > \overline{d}$. Especially, when $d_i > \overline{d}$, we need to discard $d_i^2 - \overline{d}^2$ elements in each convolution filter thus the structure and the functionality of original filters will be destroyed.

However, this size inconsistency issue can be more easily handled in the frequency domain. Resizing the DCT coefficient matrices of convolution filters in the frequency domain is reasonable, because *high-frequency* coefficients are generally small and discarding them only has a small impact on the convolution results ($d_i > \overline{d}$). On the other hand, the additionally introduced zeros will be immediately compressed by CSR since we do not need to encode or store them ($d_i < \overline{d}$). Formally, given a convolution filter F in the i-th convolutional layer, let $\mathcal{C} = \mathfrak{D}(F) \in \mathbb{R}^{d_i \times d_i}$ be its frequency coefficients, the resizing operation for convolution filters in the DCT frequency domain can be defined as:

$$\widehat{\mathcal{C}}_{j_1,j_2} = \Gamma(\mathcal{C}, \overline{d}) = \begin{cases} \mathcal{C}_{j_1,j_2}, & \text{if } j_1, j_2 \leq \overline{d}, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

where $\overline{d} \times \overline{d}$ is the fixed filter size, and $\widehat{\mathcal{C}} \in \mathbb{R}^{\overline{d} \times \overline{d}}$ is the coefficient matrix after resizing. Based on Eq. 13, we can obtain a set of DCT coefficient matrices of a given CNN with the same dimensionality. After transferring all convolution filters in a $\overline{d} \times \overline{d}$ dimensional space in the DCT frequency domain, we can pack all the coefficient matrices together and use only one set of cluster centers to compute the residual data and then compress the network. Alg. 1 summarizes the procedure of the proposed algorithm for compressing CNNs.

Algorithm 1 CNNpack for compressing deep convolutional neural networks in the frequency domain.

Input: A pre-trained convolutional neural network with p convolutional layers $\mathcal{L}_1, ..., \mathcal{L}_p$. The dimension and parameters of CNNpack: $\overline{d} \times \overline{d}$, λ, K, b and Ω.

Module 1: Filter extraction and transformation.
for each convolutional layers $\mathcal{L}_i$ in the network **do**
  for each convolution filter $F_j^{(i)}$ in $\mathcal{L}_i$ **do**
    Vectorize $F_j^{(i)}$: $\mathbf{F}^{(i)} \leftarrow [vec(F_1^{(i)}), ..., vec(F_{N_i}^{(i)})]$;
    Transfer $\mathbf{F}^{(i)}$ into the DCT frequency domain: $\mathcal{C} \leftarrow \mathbf{S}\mathbf{F}^{(i)}$ (Eq. 1);
    Resize each $\mathcal{C}_j$ in $\mathcal{C}$ to a $\overline{d} \times \overline{d}$ matrix: $\mathcal{C}_j \leftarrow \Gamma(\mathcal{C}_j, \overline{d})$ (Eq. 13);
  end for
end for
Module 2: Clustering and residual coding.
Generate K cluster centers $\mu = [\mu_1, ..., \mu_K]$ using k-means;
Shrink and quantize μ to form $\overline{\mu}$ (Eq. 10 and Eq. 11);
for each convolutional layers $\mathcal{L}_i$ in the network **do**
  for each column $\mathcal{C}_j$ in $\mathcal{C}$ **do**
    Subtract the closest center: $\mathcal{R}_j \leftarrow \mathcal{C}_j - \overline{\mu}_{j_k}$, where $\overline{\mu}_{j_k} \in \mu,\ s.t.\ \min ||\mathcal{C}_j - \overline{\mu}_{j_k}||_2$;
    Shrink the residual data $\widehat{\mathcal{R}}_j$ (Eq. 10);
    $\overline{\mathcal{R}}_j \leftarrow \mathcal{Q}\left(\widehat{\mathcal{R}}_j, \Omega, b\right)$ (Eq. 11);
  end for
end for
Module 3: Fine-tuning and compressing.
repeat
  Train the network by keeping the discarded components;
  Quantize the residual data;
until convergence
Compress $\{\overline{\mathcal{R}}_1, ..., \overline{\mathcal{R}}_p\}$ and $\overline{U}$ by exploiting CSR and Huffman encoder to form $\{\mathbf{E}_i\}$ and $\tilde{\mathbf{E}}$, respectively;

Output: The compressed data of residual data $\{\mathbf{E}_i\}$ and cluster centers $\tilde{\mathbf{E}}$, Huffman dictionaries, indexes of cluster centers.

The proposed algorithm has five hyper-parameters: λ, $\overline{d}$, K, b, and Ω. The network compression ratio can be calculated by:

$$r_c = \frac{\sum_{i=1}^{p} 32 N_i d_i^2}{\sum_{i=1}^{p} \left(N_i \log K + B_i\right) + \tilde{B} + 32 H_i}, \tag{14}$$

where p is the number of convolutional layers. It is instructive to note that a larger λ (Eq. 8) puts more emphasis on the common parts of convolution filters, which leads to a higher compression ratio r_c. A decrease in any of b, $\overline{d}$ and Ω will increase r_c accordingly. Parameter K is related to the sparseness of $\mathbf{E}_i$, and a larger K would contribute more to $\mathbf{E}_i$'s sparseness but would lead to higher storage requirement. A detailed investigation of all these parameters is presented in Section. 6, and we also demonstrate and validate the trade-off between the compression ratio and CNN accuracy of the convolutional neural work (*i.e.*, classification accuracy [41]).

4 The Data-driven Method for Compressing

Section 3 explores a compression method in the frequency domain by discarding subtle frequency coefficients. Considering the task of convolution filters to extract intrinsic patterns of input images, an ideal compression method should focus not only on the filters themselves but also on the input data. Therefore, we further extend the compression methods into a data-driven fashion.

4.1 Modeling Redundancies

For a convolutional layer $\mathcal{L}$ with filters $F = \{F_q\}_{q=1}^N$ of size $d \times d$, the input data is $X \in \mathbb{R}^{H \times W}$ and output feature maps are $Y = \{Y_q\}_{q=1}^N$, where $Y_q \in \mathbb{R}^{H' \times W'}$. We can reformulate convolutions in $\mathcal{L}$ as

$$\mathbf{Y} = \mathbf{X}^T \mathbf{F}, \tag{15}$$

where $\mathbf{Y} = [\text{vec}(Y_1), ..., \text{vec}(Y_N)] \in \mathbb{R}^{H'W' \times N}$ is a matrix which stacks all feature maps together and $\mathbf{F} = [\text{vec}(F_1), ..., \text{vec}(F_N)] \in \mathbb{R}^{d^2 \times N}$ converts all filters into a matrix similarly. $\mathbf{X}$ is a $d^2 \times H'W'$ matrix, each column of which is a $d \times d$ patch extracted from X for calculating the corresponding convolution responses in Y.

Given the data-driven motivation denoted as Eq. 15, the objective function of pruning redundancy in the frequency domain [50] is

$$\min_{\mathbf{F}} \frac{1}{2} ||\mathbf{Y} - \mathbf{X}^T \mathbf{F}||_F^2 + \lambda ||\mathbf{S}\mathbf{F}||_1, \tag{16}$$

where $\mathbf{S}$ is is the DCT transform matrix. Compared with Eq. 6, the first term of Eq. 16 can be regarded as the regression error and λ is a weight parameter for controlling the sparsity of the compressed network.

In addition, by decomposing the frequency coefficients into clusters and residuals, Eq. 16 can be rewritten as:

$$\begin{aligned} \min_{\mathcal{R}} \frac{1}{2} ||\mathbf{Y} - \mathbf{X}^T \mathbf{F}||_F^2 + \lambda ||\mathcal{R}||_1, \\ s.t. \quad \mathbf{S}\mathbf{F} = \mathcal{R} + \overline{U}, \end{aligned} \tag{17}$$

where $\overline{U}$ can be pre-obtained by using Eq. 10 and Eq. 11, and has been fixed to preserve major properties of the original network. Thus Eq. 17 can be simplified as:

$$\begin{aligned} &\min_{\mathcal{R}} \frac{1}{2} ||\mathbf{Y} - \mathbf{X}^T \mathbf{F}||_F^2 + \lambda ||\mathcal{R}||_1 \\ =&\min_{\mathcal{R}} \frac{1}{2} ||\mathbf{Y} - \mathbf{X}^T (\mathbf{S}^T (\mathcal{R} + \overline{U}))||_F^2 + \lambda ||\mathcal{R}||_1 \\ =&\min_{\mathcal{R}} \frac{1}{2} ||\tilde{\mathbf{Y}} - \mathbf{X}^T \mathbf{S}^T \mathcal{R}||_F^2 + \lambda ||\mathcal{R}||_1, \end{aligned} \tag{18}$$

where $\mathbf{S}$ is the DCT transformation matrix which is orthogonal, *i.e.*, $\mathbf{S}^T\mathbf{S} = \mathbf{I}$, and $\tilde{\mathbf{Y}} = \mathbf{Y} - \mathbf{X}^T \overline{U}$. Besides the redundancy of filters in the frequency domain, their properties in the spatial domain can also be incorporated into the formulation

$$\min_{\mathcal{R}} \frac{1}{2} ||\tilde{\mathbf{Y}} - \mathbf{X}^T \mathbf{S}^T \mathcal{R}||_F^2 + \lambda_1 ||\mathbf{S}^T \mathcal{R}||_1 + \lambda_2 ||\mathcal{R}||_1 \tag{19}$$

where $\mathbf{S}^T\mathcal{R}$ is the residual data in the spatial domain, λ_1 and λ_2 are parameters for balancing the regression error and two ℓ_1 norms. Simultaneously considering the spatial and frequency domains provides an effective approach to refine the compression performance. In practice, we set $\lambda_1 < \lambda_2$ which means that the sparsity in the spatial domain plays as an auxiliary role, since the proposed method is mainly evaluated in the frequency domain.

Since the first term in Eq. 19 encourages the feature map computed by the compressed network to be close to that of the original network, the proposed compression strategy is more focused. In specific, if we prune 90% weights solely based on Eq. 6, 20% of them might significantly impact the generated feature maps of the compressed network. However, given Eq. 19, we can remove similar amount of weights while preserving the performance of the original network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence

4.2 Optimization

There are three terms in the proposed Eq. 19, which cannot be directly optimized. To handle them independently, two auxiliary variables are introduced for help:

$$\min_{\mathcal{R}_1,\mathcal{R}_2,\mathcal{R}_3} \frac{1}{2}||\tilde{\mathbf{Y}} - \mathbf{X}^T\mathbf{S}^T\mathcal{R}_3||_F^2 + \lambda_1||\mathbf{S}^T\mathcal{R}_1||_1 + \lambda_2||\mathcal{R}_2||_1 \quad s.t. \quad \mathcal{R}_3 = \mathcal{R}_1,\ \mathcal{R}_3 = \mathcal{R}_2. \tag{20}$$

The reformulated objective function can now be more easily solved by exploiting the inexact augmented Lagrange multiplier [32]. By introducing multipliers μ_1, μ_2, E_1, E_2, the loss function can be written as

$$\begin{aligned}\mathcal{L}(\mathcal{R}_1,&\mathcal{R}_2,\mathcal{R}_3,\mu_1,\mu_2,E_1,E_2)\\ =&\frac{1}{2}||\tilde{\mathbf{Y}} - \mathbf{X}^T\mathbf{S}^T\mathcal{R}_3||_F^2 + \lambda_1||\mathbf{S}^T\mathcal{R}_1||_1 + \lambda_2||\mathcal{R}_2||_1\\ &+ < E_1, \mathcal{R}_3 - \mathcal{R}_1 > + \frac{\mu_1}{2}||\mathcal{R}_3 - \mathcal{R}_1||_F^2\\ &+ < E_2, \mathcal{R}_3 - \mathcal{R}_2 > + \frac{\mu_1}{2}||\mathcal{R}_3 - \mathcal{R}_2||_F^2.\end{aligned} \tag{21}$$

The optimal residual data $\mathcal{R}$ can be obtained by updating $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ iteratively.

Solove $\mathcal{R}_1$: The loss function w.r.t. $\mathcal{R}_1$ is

$$\begin{aligned}\mathcal{L}(\mathcal{R}_1,&\mu_1, E_1) = < \mathbf{S}^T E_1, \mathbf{S}^T(\mathcal{R}_3 - \mathcal{R}_1) >\\ &+ \frac{\mu_1}{2}||\mathbf{S}^T(\mathcal{R}_3 - \mathcal{R}_1)||_F^2 + \lambda_1||\mathbf{S}^T\mathcal{R}_1||_1,\end{aligned} \tag{22}$$

which can be simplified as

$$\begin{aligned}\mathcal{L}(\mathcal{R}_1, \mu_1, E_1) =& \frac{1}{2}||\mathbf{S}^T\mathcal{R}_1 - \mathbf{S}^T(\mathcal{R}_3 + \frac{1}{\mu_1}E_1)||_F^2\\ &+ \frac{\lambda_1}{\mu_1}||\mathbf{S}^T\mathcal{R}_1||_1.\end{aligned} \tag{23}$$

Its closed form solution is

$$\mathcal{R}_1 = \mathbf{S}\mathcal{S}_{\frac{\lambda_1}{\mu_1}}(\mathbf{S}^T\mathcal{R}_3 + \frac{1}{\mu_1}\mathbf{S}^T E_1), \tag{24}$$

where $\mathcal{S}_\lambda(x) = \text{sign}(x) \circ \max(|x| - \lambda, 0)$ is a soft-thresholding operator for solving ℓ_1 norm regularized problem.

Solove $\mathcal{R}_2$: The loss function w.r.t. $\mathcal{R}_2$ is

$$\begin{aligned}\mathcal{L}(\mathcal{R}_2, \mu_2, E_2) =& \lambda_2||\mathcal{R}_2||_1 + < E_2, \mathcal{R}_3 - \mathcal{R}_2 >\\ &+ \frac{\mu_2}{2}||\mathcal{R}_3 - \mathcal{R}_2||_F^2.\end{aligned} \tag{25}$$

Since $\mathbf{S}$ is an orthogonal matrix, *i.e.*, $\mathbf{S}^T\mathbf{S} = \mathbf{I}$, the above function can be converted into the DCT frequency domain and rewritten as

$$\begin{aligned}\mathcal{L}(\mathcal{R}_2, \mu_2, E_2) =& \lambda_2||\mathcal{R}_2||_1 + < \mathbf{S}E_2, \mathcal{R}_3 - \mathcal{R}_2 >\\ &+ \frac{\mu_2}{2}||\mathcal{R}_3 - \mathcal{R}_2||_F^2,\end{aligned} \tag{26}$$

which can be further simplified as

$$\mathcal{L}(\mathcal{R}_2, \mu_2, E_2) = \frac{\lambda_2}{\mu_2}||\mathcal{R}_2||_1 + \frac{1}{2}||\mathcal{R}_2 - (\mathcal{R}_3 + \frac{1}{\mu_2}E_2)||_F^2. \tag{27}$$

The optimal $\mathcal{R}_2$ can be obtained through the following shrinkage operation,

$$\mathcal{R}_2 = \mathcal{S}_{\frac{\lambda_2}{\mu_2}}(\mathcal{R}_3 + \frac{1}{\mu_2}E_2). \tag{28}$$

Algorithm 2 Data-driven CNNpack for compressing CNNs.

Input: A pre-trained convolutional neural network $\mathcal{N}$ with p layers: $\mathcal{L}_1, ..., \mathcal{L}_p$, and a dataset $\mathcal{X}$ for compressing, pre-trained cluster centers $\overline{U}$, parameters λ_1, λ_2, μ_1, μ_2, ρ.

Divide $\mathcal{X}$ into b batches: $\mathcal{X} = \{\mathcal{X}_1, ..., \mathcal{X}_b\}$, $\widehat{\mathcal{N}} = \mathcal{N}$;
for $i = 1$ to p **do**
  Extract convolution filters in $\mathcal{L}_i$ to form $\mathbf{F}_i$ (Eq. 15);
  $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3 = \mathbf{S}\mathbf{F}_i - \overline{U}$ (Eq. 17);
  $E_1 = E_2 = \mathcal{R}_1/J(\mathcal{R}_1)$, $\mu_1 > 0$, $\mu_2 > 0$, $\rho > 1$;
  repeat
    Randomly select a batch $\mathcal{X}_j$ from $\mathcal{X}$;
    Calculate the input data X_i of $\mathcal{L}_i$ by using $\widehat{\mathcal{N}}$;
    Calculate feature maps Y_i of $\mathcal{L}_i$ by using $\mathcal{N}$;
    Form $\mathbf{X} \leftarrow X_i$, $\mathbf{Y} \leftarrow Y_i$, $\tilde{\mathbf{Y}} = \mathbf{Y} - \mathbf{X}^T U$ (Eq. 18);
    $\mathcal{R}_1 \leftarrow \mathbf{S}\mathcal{S}_{\frac{\lambda_1}{\mu_1}}(\mathbf{S}^T\mathcal{R}_3 + \frac{1}{\mu_1}\mathbf{S}^T E_1)$;
    $\mathcal{R}_2 \leftarrow \mathcal{S}_{\frac{\lambda_2}{\mu_2}}(\mathcal{R}_3 + \frac{1}{\mu_2}E_2)$;
    $G \leftarrow \mathbf{S}\mathbf{X}\tilde{\mathbf{Y}} - E_1 - E_2 + \mu_1\mathcal{R}_1 + \mu_2\mathcal{R}_2$;
    $\mathcal{R}_3 \leftarrow (\mathbf{S}\mathbf{X}\mathbf{X}^T\mathbf{S}^T + \mu_1\mathbf{I} + \mu_2\mathbf{I})^{-1}G$;
    $E_1 \leftarrow E_1 + \mu_1(\mathcal{R}_3 - \mathcal{R}_1)$;
    $E_2 \leftarrow E_2 + \mu_2(\mathcal{R}_3 - \mathcal{R}_2)$;
    $\mu_1 \leftarrow \rho\mu_1$, $\mu_2 \leftarrow \rho\mu_2$;
  until convergence
  Quantize $\mathcal{R}_3$: $\overline{\mathcal{R}} \leftarrow \mathcal{Q}(\mathcal{R}_3, \Omega, b)$;
  Calculate the optimal filter matrix: $\widehat{\mathbf{F}} \leftarrow \mathbf{S}^T(\overline{\mathcal{R}} + U)$;
  Embed $\widehat{\mathbf{F}}$ into the convolution layer $\mathcal{L}_i$ in $\widehat{\mathcal{N}}$;
end for
Fine-tune $\widehat{\mathcal{N}}$ by keeping the discarded components;

Output: The new convolutional neural network $\widehat{\mathcal{N}}$.

Solove $\mathcal{R}_3$: The loss function w.r.t. $\mathcal{R}_3$ is

$$\begin{aligned}\mathcal{L}(\mathcal{R}_3, \mu_1, \mu_2, &E_1, E_2) = \frac{1}{2}||\tilde{\mathbf{Y}} - \mathbf{X}^T\mathbf{S}^T\mathcal{R}_3||_F^2 +\\ &+ < E_1, \mathcal{R}_3 - \mathcal{R}_1 > + \frac{\mu_1}{2}||\mathcal{R}_3 - \mathcal{R}_1||_F^2\\ &+ < E_2, \mathcal{R}_3 - \mathcal{R}_2 > + \frac{\mu_1}{2}||\mathcal{R}_3 - \mathcal{R}_2||_F^2.\end{aligned} \tag{29}$$

By minimizing its gradient, we can obtain the optimal solution of $\mathcal{R}_3$ as

$$\mathcal{R}_3 = (\mathbf{S}\mathbf{X}\mathbf{X}^T\mathbf{S}^T + \mu_1\mathbf{I} + \mu_2\mathbf{I})^{-1}G, \tag{30}$$

where $G = \mathbf{S}\mathbf{X}\mathbf{Y} - E_1 - E_2 + \mu_1\mathcal{R}_1 + \mu_2\mathcal{R}_2$. Finally, multipliers are updated according

$$E_1 = E_1 + \mu_1(\mathcal{R}_3 - \mathcal{R}_1), \quad E_2 = E_2 + \mu_2(\mathcal{R}_3 - \mathcal{R}_2), \quad \mu_1 = \rho\mu_1, \quad \mu_2 = \rho\mu_2, \tag{31}$$

where $\rho > 1$ is a user-defined constant, and the optimal filter matrix can be obtained as

$$\widehat{\mathbf{F}} = \mathbf{S}^T\left(\overline{\mathcal{R}} + \overline{U}\right), \tag{32}$$

where $\overline{\mathcal{R}}$ is the quantized residual data of $\mathcal{R}_3$. Since the dataset for training a sophisticated CNN usually has more than 100 thousands samples, *e.g.*, ImageNet dataset [40], tiny images dataset [45], we use the mini-batch approach [35], [15] to optimize the proposed compression method as shown in Alg. 2.

Moreover, since the learned network $\widehat{\mathcal{N}}$ is sparse in the frequency domain, we use the Huffman encoding and CSR to further compress it after quantizing by employing Eq. 13, so that the compression ratio could achieve the result in Eq. 14.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence

5 Speeding Up Convolutions

In above sections, we have proposed effective algorithms for learning compact models of pre-trained convolutional neural networks in the frequency domain. According to Eq. 14, we can obtain considerable compression ratios by converting the convolution filters into the DCT frequency domain and representing them using frequency coefficients.

If online inference is executed in the spatial domain as usual, frequency representations of filters have to be transformed back to the spatial domain using the inverse DCT (Eq. 4), *i.e.*, decompressing the compressed data $\mathbf{E}$. Even if there is only one non-zero component in the frequency domain, its inverted data in the spatial domain are dense valued. The online memory thus cannot be really saved, and the computational complexity of the convolutions will be the same as that of the original network, let alone the extra transformation cost from frequency to spatial domain. Hence, we present a novel convolution method in the DCT frequency domain, where both original input data and convolution filters are represented in the frequency domain.

Given a convolutional layer $\mathcal{L}$ with filters $F = \{F_q\}_{q=1}^N$ of size $d \times d$, we denote the input data (image or feature map) as $X \in \mathbb{R}^{H \times W}$ and its output feature maps as $Y = \{Y_1, Y_2, ..., Y_N\}$ with size $H' \times W'$, where $Y_q = F_q * X$ under the convolution operation $*$. For the DCT matrix $C = [c_1, ..., c_d]$ (Eq. 1), the $d \times d$ convolution filter F_q can be represented by its DCT coefficient matrix $\mathcal{C}^{(q)}$ with DCT bases $\{S_{j_1,j_2}\}_{j_1,j_2=1}^d$ defined as $S_{j_1,j_2} = c_{j_1} c_{j_2}^T$, namely,

$$F_q = \sum_{j_1=1}^{d} \sum_{j_2=1}^{d} \mathcal{C}_{j_1,j_2}^{(q)} S_{j_1,j_2}. \tag{33}$$

In this way, feature maps of X through F can be calculated as

$$Y_q = \sum_{j_1,j_2=1}^{d} \mathcal{C}_{j_1,j_2}^{(q)} (S_{j_1,j_2} * X), \tag{34}$$

where M is the number of DCT bases. Based on Eq. 34, for the input data (feature maps generated by the previous layer or the input image of the first convolutional layer) X, we first transform it into the DCT frequency domain by calculating its responses on DCT bases and then generate feature maps using frequency coefficients of filters.

As for the speed-up ratio r_s, it is obvious that $r_s > 1$ only when $M \ll N$. Since $M = d^2$ in the DCT, Eq. 34 cannot be utilized for speeding up CNNs effectively when a layer has few filters.

But considering the fact that the DCT is an orthogonal transformation and all of its bases are rank-1 matrices, we thus have $S_{j_1,j_2} * X = \left(c_{j_1} c_{j_2}^T\right) * X$. The feature map Y_q can then be re-written as

$$\begin{aligned} Y_q = F_q * X &= \sum_{j_1,j_2=1}^{d} \mathcal{C}_{j_1,j_2}^{(q)} (S_{j_1,j_2} * X) \\ &= \sum_{j_1,j_2=1}^{d} \mathcal{C}_{j_1,j_2}^{(q)} [c_{j_1} * (c_{j_2}^T * X)]. \end{aligned} \tag{35}$$

The above function can reduce the computational complexity of the convolution of a DCT basis from $\mathcal{O}(d^2)$ to $\mathcal{O}(2d)$, which also needs to be further squeezed.

Revisiting the DCT, for a given $d \times d$ matrix, we can obtain d^2 frequency coefficients by only applying the DCT once. Thus it is worth for us to explore the relationship between the DCT coefficients and the convolutional responses of the DCT bases $\{S_{j_1,j_2}\}$. DCT can be regarded as a linear decomposition by using its fixed bases, whose frequency components are exactly the convolution responses of its bases, as proved in Theorem 1.

Theorem 1. *For a $d \times d$ matrix X, its DCT coefficients are calculated as $\mathcal{C} = C^T X C$, where $\mathcal{C}_{j_1,j_2}$ is the frequency coefficient corresponding to the basis S_{j_1,j_2}. $\mathcal{C}_{j_1,j_2}$ is also exactly the convolution response of S_{j_1,j_2} to X.*

Proof. The DCT basis S_{j_1,j_2} can be calculated as a convolution of two cosine bases c_{j_1} and c_{j_2}, *i.e.*

$$S_{j_1,j_2} = c_{j_1} c_{j_2}^T = c_{j_1} * c_{j_2}^T, \tag{36}$$

and the calculation of its corresponding frequency coefficient $\mathcal{C}_{j_1,j_2}$ can be rewritten as

$$\begin{aligned} \mathcal{C}_{j_1,j_2} &= c_{j_1}^T X c_{j_2} = c_{j_1}^T (c_{j_2}^T * X) \\ &= c_{j_1} * (c_{j_2}^T * X) = (c_{j_1} * c_{j_2}^T) * X \\ &= (c_{j_1} c_{j_2}^T) * X = S_{j_1,j_2} * X, \end{aligned} \tag{37}$$

thus the frequency component $\mathcal{C}_{j_1,j_2}$ of X is equal to its convolution response obtained by S_{j_1,j_2}. □

According to Theorem 1, it is encouraging that the conventional convolutions of DCT bases can be accelerated significantly. Moreover, beneficial from the proposed compression scheme in Alg. 1, any $\mathcal{C}^{(q)}$ in Eq. 34 is extremely sparse. Thus the computational complexity of our proposed scheme can be further reduced, as analyzed in Proposition 1.

Proposition 1. *Given a convolutional layer with N filters, and filter size is $d \times d$. $M = d \times d$ is the number of DCT base, and $\mathcal{C} \in \mathbb{R}^{d^2 \times N}$ denote the frequency coefficients of filters in this layer. Suppose δ is the ratio of non-zero elements in $\mathcal{C}$, while η is the ratio of non-zero elements in K' active cluster centers of this layer. The computational complexity of our proposed scheme is $\mathcal{O}((d^2 \log d + \eta M K' + \delta M N) H' W')$ for calculating convolutions of this layer.*

Proof. The computational complexity for the feature maps Y can be computed as $\mathcal{O}(d^2 N H' W')$. When implementing the compressed CNN with our proposed algorithm, a naive approach would be to invert all frequency-filters into the spatial domain and then calculate spatial convolutions. Since the computational complexity of a $d \times d$ DCT is $\mathcal{O}(d^2 \log d)$ [1], the overall complexity of the method will be

$$\mathcal{O}(d^2 \log d N + d^2 N H' W') \tag{38}$$

or, equally, $\mathcal{O}(d^2 N H' W')$ considering $d^2 \log d N \ll d^2 N H' W'$. However, this simple method tends to be inefficient since it involves a lot of redundant computation. Hence, we propose to first use the DCT bases as a set of filter bases to obtain a set of feature maps; the feature map of a convolution filter can then be quickly calculated by summarizing them based on their DCT coefficients $\mathcal{C}$.

Since we decompose the traditional convolutions by combinations of feature maps of DCT bases in Eq. 35, the complexity should be rewritten as

$$\mathcal{O}((2dM + MN) H' W'). \tag{39}$$

Algorithm 3 CNNpack for speeding up deep convolutional neural networks in the frequency domain.

Input: A compressed convolutional layer $\mathcal{L}$ with N filters $F = \{F_1, ..., F_N\}$, filter size is $d \times d$, input data X, and the size of feature maps of this layer is $H' \times W'$. The pre-operated cluster centers $\overline{\mu}$.

1: Divide X into $H' \times W'$ patches with size $d \times d$: $\{X^{(i)}\}$;
2: **for** each local region $X^{(i)}$ in X **do**
3: Calculate M DCT coefficients of X applying Eq. 1 and Eq. 37: $\mathcal{C}_{j_1,j_2} \leftarrow S_{j_1,j_2} * X^{(i)}, \forall j_1, j_2 = 1, ..., d$;
4: Calculate feature maps of cluster centers: $Y_{\overline{\mu}} \leftarrow \sum_{j_1,j_2=1}^{d} \overline{\mu}_{j_1,j_2}(S_{j_1,j_2} * X^{(i)})$;
5: **for** each convolution filter in $\mathcal{L}$ **do**
6: Calculate the convolution of its private part: $Y_q^{(i)} \leftarrow \sum_{j_1,j_2=1}^{d} \overline{\mathcal{R}}_{j_1,j_2}(S_{j_1,j_2} * X^{(i)})$;
7: Calculate the convolution: $Y_q^{(i)} \leftarrow Y_q^{(i)} + Y_{\overline{\mu}_k}^{(i)}$; where $\overline{\mu}_k \in U,\ s.t.\ \min ||\mathcal{C}_j - \overline{\mu}_k||_2$
8: **end for**
9: **end for**

Output: Feature maps of $\mathcal{L}$ through convolution filters F: $Y = \{Y_1, Y_2, ..., Y_N\}$ with size $H' \times W'$.

Moreover, feature maps of those DCT bases can be quickly calculated by using Eq. 37, and the complexity of a $d \times d$ DCT is only $\mathcal{O}(d^2 \log d)$ [1]. Thus the complexity for calculating feature maps of DCT bases is $\mathcal{O}(d^2 \log d)$, and the complexity of Eq. 35 is reduced to

$$\mathcal{O}((d^2 \log d + MN)H'W'). \tag{40}$$

Furthermore, if the compressed residual data of convolution filters $\overline{\mathcal{R}}$ is sufficiently sparse, the complexity can be rewritten as

$$\mathcal{O}((d^2 \log d + \sum_{i=1}^{N} ||\overline{\mathcal{R}}||_0)H'W'). \tag{41}$$

It is important to note that the complexity of Eq. 41 would be significantly smaller than the original complexity given $\sum_{i=1}^{N} ||\overline{\mathcal{R}}||_0 \ll MN$. We can simplify it as $\sum_{i=1}^{N} ||\overline{\mathcal{R}}||_0 = \delta MN$, where δ is a small value (*e.g.*, $\delta = 0.05$) denoting the sparse degree of the compressed CNN. A stronger sparseness penalty would encourage δ to be smaller.

Moreover, we will need to calculate the feature maps of cluster centers. Since U has been obtained over all convolution filters in different layers in a network, if the convolution filters in a layer correspond to only K' centers (where $K' \leq K$), additional computational cost will be saved in this layer thus the complexity can be written as

$$\begin{aligned}\mathcal{O}((d^2 \log d + \sum_{k=1}^{K'} ||U'_k||_0)H'W') = \\ \mathcal{O}((d^2 \log d + \eta MK')H'W'),\end{aligned} \tag{42}$$

where η is similar to δ and denotes the sparse degree of cluster centers U' for this layer. Since we only need to calculate the feature maps of DCT bases once, the complexity of our proposition is

$$\mathcal{O}((d^2 \log d + \eta MK' + \delta MN)H'W'), \tag{43}$$

which is smaller than that ($\mathcal{O}(d^2 NH'W')$) of the original convolutions when η and δ are both $\ll 1$. □

According to Proposition 1, the proposed compression scheme in the DCT frequency domain can improve the speed of CNNs significantly due to both η and δ are extremely small after removing redundancies. Compared to the original CNN, for a convolutional layer, the speed-up of Alg. 1 is

$$r_s = \frac{d^2 NH'W'}{(d^2 \log d + \eta K'M + \delta NM)H'W'} \approx \frac{N}{\eta K' + \delta N}. \tag{44}$$

Obviously, the speed-up ratio of the proposed method is directly relevant to η and δ, which correspond to λ (Eq. 8), λ_1 and λ_2 (Eq. 19). Alg. 3 summarizes the detailed procedures of the proposed scheme for calculating feature maps of a convolutional layer. By the way, the fast Fourier transform (FFT) also be studied for speeding up CNNs [39], [54], wherein, the convolution theorem was utilized for calculating convolutions. But Fourier coefficients are imaginary numbers which are not conductive for compressing and the feature maps calculated in the frequency domain need to be converted into the spatial domain for subsequent operations such as pooling and ReLu.

6 Experimental Results

Baselines and Models. We compared the proposed algorithms with several baseline approaches: Perforation [16], P+QH (Pruning + Quantization and Huffman encoding) [19], SVD [13], XNOR-Net [36], and LCNN [3]. The evaluation was conducted using the MNIST and ILSVRC2012 datasets. We evaluated the proposed compression approach over four widely used CNNs: LeNet [31], [47], AlexNet [29], VGG-16 Net [41], ResNet-50 [23], and ResNeXt-50 (32 × 4d) [53]. All methods were implemented using MatConvNet [47] and run on K40 graphics cards. Model parameters were stored and updated as 32 bit floating-point values.

Impact of parameters. As discussed above, the proposed compression method has several important parameters: λ, $\overline{d}$, K, b, and Ω. We first tested their impact on the network accuracy of a LeNet by conducting an experiment using MNIST [47], where the network has two convolutional layers and two fully-connected layers of sizes $5 \times 5 \times 1 \times 20$, $5 \times 5 \times 20 \times 50$, $4 \times 4 \times 50 \times 500$, and $1 \times 1 \times 500 \times 10$, respectively. The original model accuracy was 99.06%. The compression results of different λ and $\overline{d}$ after fine-tuning are shown in Fig. 3, wherein, k was set as 16, b was equal to $+\infty$ since it did not make an obvious contribution to the compression ratio even when set at a relatively smaller value (*e.g.*, $b = 0.05$) but caused the accuracy reduction. Ω was set to 500, making the average length of weights in the frequency domain about 6, a bit larger than that in [19] but more flexible and with relatively better performance. Note that all the training parameters used their default settings, such as epochs, learning rates, *etc*.

It can be seen from Fig. 3 that although a lower $\overline{d}$ slightly improves the compression ratio and speed-up ratio simultaneously, this comes at a cost of decreased overall network accuracy; thus, we kept $\overline{d} = \max\{d_i\},\ \forall\ i = 1, ..., p$, in CNNpack. Overall, λ is clearly the most important parameter in the proposed scheme, which is sensitive but monotonous. Thus, it only needs to be adjusted according to demand and restrictions. Furthermore, we tested the impact of number of cluster centers K. As mentioned above, K is special in that its impact on performance is not intuitive. When K becomes larger, $\mathbf{E}$ becomes sparser but more space is required for storing cluster centers U and indexes. Fig. 4 shows

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence

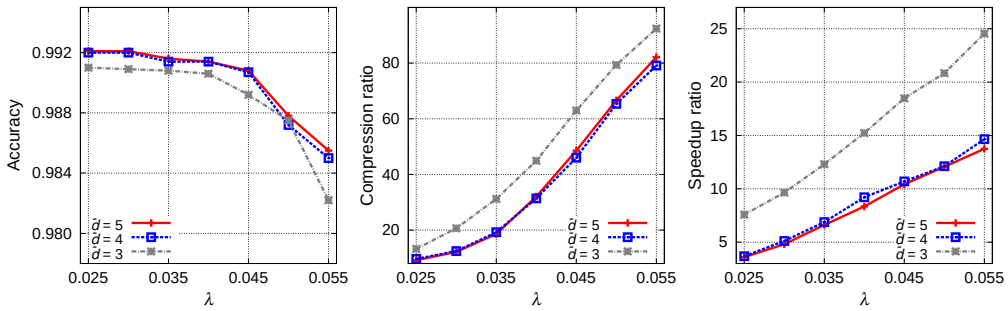


Fig. 3: The performance of the proposed approach with different λ and $\overline{d}$.

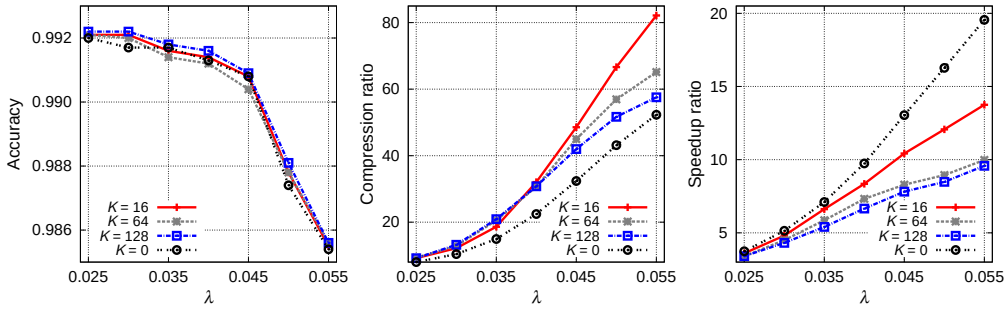


Fig. 4: The performance of the proposed approach with different numbers of cluster centers K.

that $K = 16$ provides the best trade-off between compression performance and accuracy.

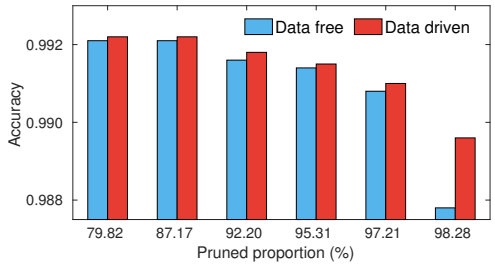


Fig. 5: Comparison between data-free and data-driven methods by pruning different proportions of weights on MNIST.

We also report the compression results by directly compressing the DCT frequency coefficients of original filters $\mathcal{C}$ as before (*i.e.*, $K = 0$, the black line in Fig. 4). It can be seen that the clustering number does not affect accuracy, but a suitable K does enhance the compression ratio. Another interesting phenomenon is that the speed-up ratio without decomposition is larger than that of the proposed scheme because the network is extremely small and the clustering introduces additional computational cost as shown in Eq. 44. However, recent networks contain a lot more filters in a convolutional layer, larger than $K = 16$. Based on the above analysis, we kept $\lambda = 0.04$ and $K = 16$ for this network (an accuracy of 99.14%). Accordingly, the compression ratio $r_c = 32.05\times$ and speed-up ratio $r_s = 8.34\times$, which is the best trade-off between accuracy and compression performance.

Data-free v.s. Data-driven. In addition, a data-driven method for compressing CNNs has been proposed in Alg. 2, which can provide a more accurate guidance for removing redundant weights in CNNs. In order to illustrate its superiority, we compared its performance with that of the original data-free method by ranging the pruned proportion of all weights in the network as shown in Fig. 5.

As can be found in Fig. 5, the data-driven method can hold a higher accuracy when pruning similar amounts of weights in the network. Although the data-driven method needs more computation times for learning the sparse network in the frequency domain, but these are off-line computations. As a result, we obtained a $35.42\times$ compression ratio with an accuracy of 99.15%, which is about $2\times$ higher than that of the data-free method. The speed-up ratio of the new method is $8.59\times$, which is also higher than that of the data-free method.

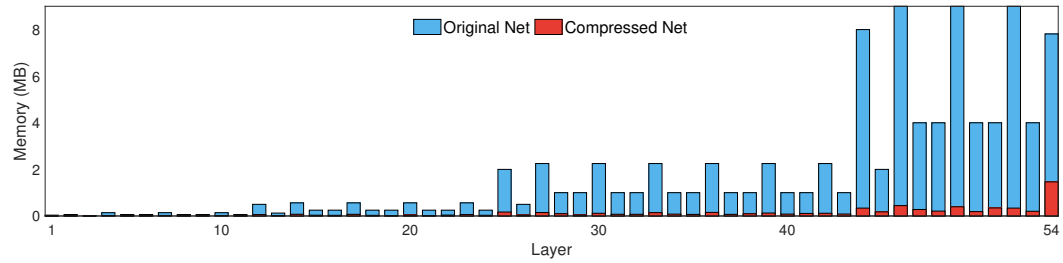
Extensive ablation experiments. The proposed CNNpack algorithm consists of several essential components, (*i.e.*, (s_1) DCT transformation, (s_2) k-means clustering, (s_3) ℓ_1-regularization, (s_4) quantization, (s_5) Huffman coding, and (s_6) CSR format, as shown in the top line in Fig. 2. Wherein, the influence of s_2 has been fully investigated in Fig. 4, and the number of cluster centers was set as $K = 16$ for an appropriate trade-off between compression and speed-up performance. In fact, independently exploiting s_1 has no influence on the compression, since DCT is a lossless and linear transform; s_3 and s_6 have to be considered together since CSR is only effective for sparse data; and s_4 is usually launched before s_5 for the convenience of coding. Hence, we proceed to evaluate the performance of Comb-1 (s_1, s_2, s_3, and s_6) and Comb-2 (s_1, s_2, s_4, and s_5) on the MNIST dataset.

TABLE 1: Comparison between combinations of different components in the proposed algorithm.

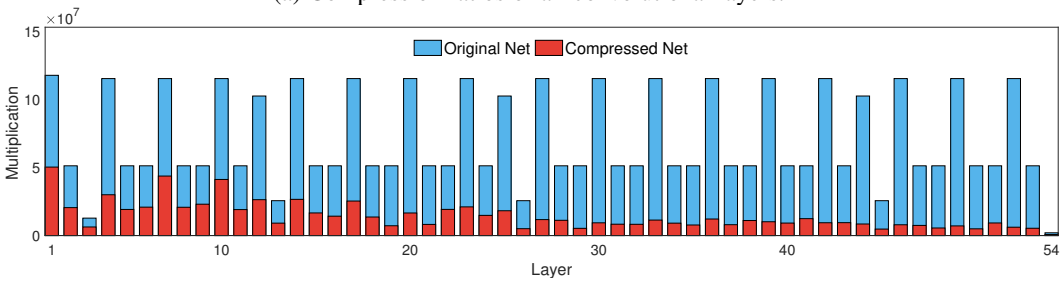
Performance	Comb-1	Comb-2	CNNpack
r_c	$13.45\times$	$5.32\times$	$35.42\times$
r_s	$8.59\times$	$1\times$	$8.59\times$

It can be found in Tab. 1 that, the speed-up benefit of the proposed CNNpack was obtained from Comb 1, because of the acceleration of multiplications using sparse DCT frequency coefficients of compressed filters by Eq. 34. In addition, given the sparse representation and the CSR format, Comb1 brought in a $13.45\times$ compression ratio. Obviously, the proposed CNNpack can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence



(a) Compression ratios of all convolutional layers.



(b) Speed-up ratios of all convolutional layers.

Fig. 6: Compression statistics for ResNet-50 (better viewed in color version).

achieve a higher compression ratio by combining the quantization and Huffman encoding in Comb 2.

Filter visualization. The proposed algorithm operates in the frequency domain. Though we do not need to transform the compressed net back into the spatial domain when calculating convolutions, we reconstruct the convolution filters in the spatial domain for a more intuitive visualization. Reconstructed convolution filters are obtained from the LeNet on MNIST, as shown in Fig. 7.

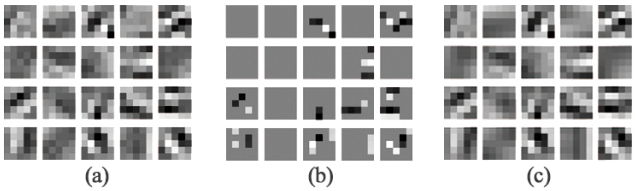


Fig. 7: Visualization of example filters learned on MNIST: (a) the original convolution filters, (b) filters after pruning, (c) convolution filters compressed by the proposed algorithm.

The proposed approach is fundamentally different to the previously used pruning algorithm. According to Fig. 7(b), weights with smaller magnitudes are pruned while influences of larger weights have been discarded. In contrast, our proposed algorithm not only handles the smaller weights but also considers impacts of those larger weights. Most importantly, we accomplish the compressing task by exploring the underlying connections between all the weights in the convolution filter (see Fig. 7(c)).

Compression CNNs on ImageNet. We next employed CNNpack for CNN compression on the ImageNet ILSVRC-2012 dataset [40], which contains over 1.2M training images and 50k validation images. First, we examined two conventional models: AlexNet [29], with over 61M parameters and a top-5 accuracy of 80.8%; and VGG-16 Net, which is much larger than the AlexNet with over 138M parameters and has a top-5 accuracy of 90.1%. Table 2 shows detailed compression and speed-up ratios of the AlexNet with $K = 16$. The result of the VGG-16 Net with the same setting can be found in Table 3. The reported multiplications are for computing one image.

TABLE 2: Compression statistics for AlexNet.

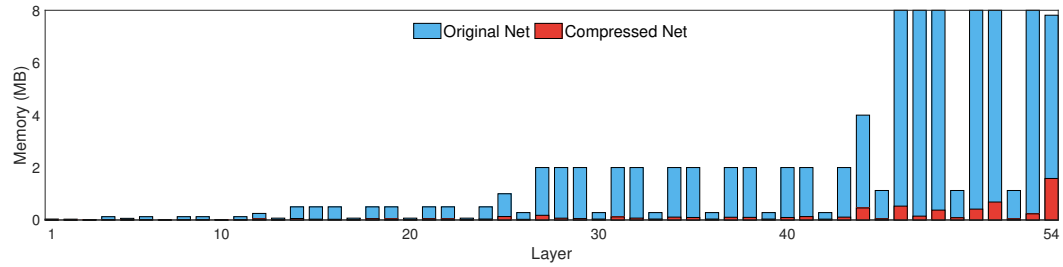
Layer	Memory	r_c	Multiplication	r_s
conv1	0.13MB	868×	1.05×10^8	110×
conv2	1.17MB	124×	2.23×10^8	30×
conv3	3.37MB	949×	1.49×10^8	29×
conv4	2.53MB	65×	1.12×10^8	18×
conv5	1.68MB	60×	0.74×10^8	13×
fc6	144MB	358×	0.37×10^8	216×
fc7	64MB	16×	0.16×10^8	8×
fc8	15.62MB	121×	0.04×10^8	60×
Total	232.52MB	43.5×	7.24×10^8	26.2×

TABLE 3: Compression statistics for VGG-16 Net.

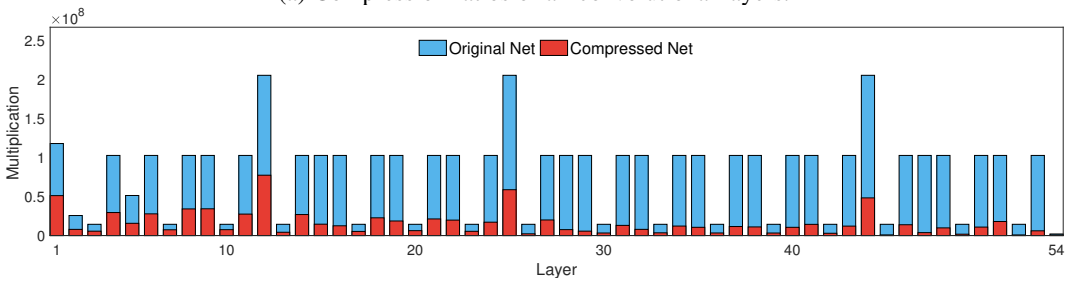
Layer	Memory	r_c	Multiplication	r_s
conv1_1	0.006MB	302×	0.11×10^9	35×
conv1_2	0.14MB	28×	2.41×10^9	8×
conv2_1	0.28MB	15×	1.20×10^9	6×
conv2_2	0.56MB	16×	2.41×10^9	7×
conv3_1	1.12MB	18×	1.20×10^9	9×
conv3_2	2.25MB	16×	2.41×10^9	8×
conv3_3	2.25MB	32×	2.41×10^9	14×
conv4_1	4.5MB	14×	1.20×10^9	8×
conv4_2	9MB	47×	2.41×10^9	24×
conv4_3	9MB	54×	2.41×10^9	27×
conv5_1	9MB	9×	0.60×10^9	6×
conv5_2	9MB	14×	0.60×10^9	9×
conv5_3	9MB	25×	0.60×10^9	15×
fc6	392MB	270×	0.41×10^9	201×
fc7	64MB	15×	0.16×10^8	8×
fc8	15.62MB	215×	0.41×10^7	120×
Total	572.74MB	49.1×	2.04×10^{10}	10.2×

We achieved a 43.5× compression ratio and a 49.1× compression ratio for AlexNet and VGG-16 Net, respectively. In contrast, compression ratios of the data-free method for these two networks are 39.3× and 46.2×, respectively. The layer with a relatively larger filter size has a larger compression ratio because it contains more subtle high-frequency coefficients. In contrast, the highest speed-up ratio is often obtained on the layer whose filter number N was much larger than its filter size, *e.g.*, the fc6 layer of AlexNet. We only obtained a 10.2× speed-up ratio on VGG-16

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2857824, IEEE Transactions on Pattern Analysis and Machine Intelligence



(a) Compression ratios of all convolutional layers.



(b) Speed-up ratios of all convolutional layers.

Fig. 8: Compression statistics for ResNeXt-50 (32 × 4d) (better viewed in color version).

Net because the layer complexity is relevant to the feature map size and the first several layers definitely have more multiplications. Unfortunately, their filter numbers are relatively small and their compression ratios are all small, thus the overall speed-up ratio is lower than that on AlexNet. Accordingly, when we set $K = 0$, the compression ratio and the speed-up ratio of AlexNet were $35\times$ and $22\times$ and those of VGG-16 Net were $28\times$ and $7.05\times$. This reduction is because that these two networks are relatively large and contain many similar filters. Moreover, the filter number in each layer is larger than the number of cluster centers, *i.e.*, $N > K$. Thus, cluster centers can effectively reduce memory consumption and computational complexity simultaneously.

ResNet-50 and ResNeXt-50 on ImageNet. Here we discuss a more recent work, ResNet-50 [23], which has more than 150 layers and 54 convolutional layers. This model achieves a top-5 accuracy of 7.71% and a top-1 accuracy of 24.62% with only about 95MB parameters [47]. Moreover, since this model adopts small filters with sizes 1×1, 3×3, and 7×7, it is harder to launch compression on the ResNet-50 compared with traditional AlexNet and VGGNet.

For the experiment on ResNet-50, we set $K = 0$ since the functionality of quantization (Eq. 11) for 1-dimensional filters is similar to that of k-means clustering, and cluster centers are dispensable for these models. We obtained a 7.8% top-5 accuracy on the ResNet-50. Fig. 6 shows detailed compression statistics of ResNet-50 utilizing the proposed CNNpack and CNNpack v2 (the proposed data-driven method as detailed Alg. 2). In summary, memory usage for storing filters of the ResNet-50 was squeezed by a factor of $14.0\times$, and the speed-up ratio for this network is about $5.0\times$.

Compared with results on AlexNet and VGGNet-16, compression and speed-up ratios on the ResNet-50 are obviously lower since the ResNet-50 has a more compact architecture by utilizing convolution filters with small sizes, *i.e.*, 7×7, 3×3, and 1×1. Although modern CNNs adopt small filters, they still have a lot of convolutional layers with larger filters of sizes 7×7 and 3×3, which accounts for more than half proportion of those of whole networks (*e.g.*, ResNet [23] and ResNeXt [53]). Therefore, it is reasonable to compress CNNs in the DCT frequency domain, as discussed in Fig. 1.

TABLE 4: Compression statistics for ResNets.

Model	Evaluation	Original	CNNpack	CNNpack v2
ResNet-50 [23]	r_c	1	12.3×	14.0×
	r_s	1	4.4×	5.0×
	top-1 *err*	24.6%	24.8%	24.7%
	top-5 *err*	7.7%	7.8%	7.8%
ResNeXt-50 [53]	r_c	1	12.6×	14.3×
	r_s	1	4.5×	5.1×
	top-1 *err*	22.6%	23.8%	23.6%
	top-5 *err*	6.5%	6.9%	6.8%

In addition, we also tested the performance of the proposed CNNpack on the ResNeXt-50 [53] which enhances ResNet-50 by dividing conventional convolutional layers into a set of small layers. This model achieves a top-5 accuracy of 6.52% and a top-1 accuracy of 22.67% with a similar architecture and number of weights to those of the original ResNet-50. After applying the proposed compression scheme, we obtained a $14.3\times$ compression ratio and a $5.1\times$ speed-up ratio on the ResNeXt-50, and detailed compression statistics on this network are also shown in Fig. 8. Tab. 4 summarizes compression results on ResNet-50 and ResNeXt-50, respectively. It is clear that the proposed CNNpack is applicable to this recent network with compact architecture, since there are still many 3×3 convolution filters in this model.

Comparison with state-of-the-art methods. We detail a comparison with state-of-the-art methods for compressing CNNs in Tab. 5. CNNpack clearly achieves the best performance in terms of both the compression ratio (r_c) and the speed-up ratio (r_s). Note that although Pruning+QH achieves a similar compression ratio to the proposed method, the data in their algorithm is stored after applying encoding, which means that filters have to be decoded before any calculation. Hence, the compression ratio of P+QH will be lower than that reported in [19] if we only consider memory usage. In contrast, the compressed data produced by our method can be directly used for network calculation. In reality, online memory usage is the real restriction for mobile devices, and the proposed method is superior to previous works in terms of both

TABLE 5: An overall comparison of state-of-the-art methods for deep neural network compression and speed-up on the ILSVRC2012 dataset, where r_c is the compression ratio and r_s is the speed-up.

Model	Evaluation	Original	Perforation [16]	P+QH [19]	SVD [13]	XNOR [36]	LCNN [3]	CNNpack	CNNpack v2
AlexNet [29]	r_c	1	1.7×	35×	5×	32×	-	39.3×	43.5×
	r_s	1	2×	-	2×	8×	37.6×	25.1×	26.2×
	top-1 err	41.8%	44.7%	42.7%	44.0%	56.8%	55.7%	41.6%	41.9%
	top-5 err	19.2%	-	19.7%	20.5%	31.8%	31.3%	19.2%	19.3%
VGGNet-16 [41]	r_c	1	1.7×	49×	-	-	-	46.2×	49.1×
	r_s	1	1.9×	3.5×	-	-	-	9.4×	10.2×
	top-1 err	28.5%	31.0%	31.1%	-	-	-	29.7%	29.4%
	top-5 err	9.9%	-	10.9%	-	-	-	10.4%	10.2%

the compression ratio and the speed-up ratio.

Running time. We reported CNN runtimes before and after applying the proposed method in Tab. 6. Original and compressed models were both launched using MatConvNet [47] and NVIDIA K40 cards. It can be seen that the running time of compressed model was significantly reduced. The practical speed-up ratio was slightly lower than the theoretical speed-up ratio r_s due to costs incurred by data transmission, pooling, padding, *etc*.

TABLE 6: Running time of different networks per image.

Time	Original	Compressed	speed-up
AlexNet	1.82 ms	0.09 ms	20.5×
VGGNet-16	16.67 ms	2.34 ms	7.1×
ResNet-50	9.03 ms	2.89ms	3.1×
ResNeXt-50	9.56 ms	3.27ms	2.9×

7 Conclusion

Neural network compression techniques are desirable so that CNNs can be used on mobile devices. Therefore, here we present an effective compression scheme in the DCT frequency domain, namely, CNNpack. Compared to state-of-the-art methods, we tackle this issue in the frequency domain, which can offer the probability for more compression ratio and speed-up. Moreover, we no longer independently consider each weight since each frequency coefficients calculation involves all weights in the spatial domain. Following the proposed compression approach, we explore a much cheaper convolution calculation based on the sparsity of the compressed net in the frequency domain. Although the compressed network produced by our approach is sparse in the frequency domain, the compressed model has the same functionality as the original network since filters in the spatial domain have preserved intrinsic structure. In addition, we extended the proposed method into a data-driven method, which allows us to discard more useless weights in deep models. Our experiments show that the compression ratio and the speed-up ratio are both higher than those of state-of-the-art methods. The proposed CNNpack approach creates a bridge to link traditional signal and image compression with CNN compression theory, allowing us to further explore CNN approaches in the frequency domain.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant NSFC 61375026 and 2015BAF15B00, and Australian Research Council Projects: FT-130101457, DP-140102164, LP-150100671.

References

[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *Computers, IEEE Transactions on*, 100(1):90–93, 1974.
[2] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable bounds for learning some deep representations. *ICML*, 2014.
[3] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. In *CVPR*, 2017.
[4] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.
[5] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE TPAMI*, 35(8):1798–1828, 2013.
[6] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.
[7] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks. *arXiv preprint arXiv:1506.04449*, 2015.
[8] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.
[9] Y.-N. Chen, C.-C. Han, C.-T. Wang, B.-S. Jeng, and K.-C. Fan. A cnn-based face detector with a simple feature map and a coarse-to-fine classifier-withdrawn. *IEEE TPAMI*, 2009.
[10] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.
[11] M. Courbariaux, Y. Bengio, and J.-P. B. David. Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.
[12] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *NIPS*, 2013.
[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.
[14] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE TPAMI*, 38(2):295–307, 2016.
[15] J. Feng, H. Xu, and S. Yan. Online robust pca via stochastic optimization. In *NIPS*, 2013.
[16] M. Figurnov, D. Vetrov, and P. Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. *NIPS*, 2016.
[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.
[18] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.
[19] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.
[20] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.
[21] S. J. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In *NIPS*, 1989.
[22] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: optimal brain surgeon. In *NIPs*, 1993.
[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.
[24] D. A. Huffman et al. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
[25] K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *IEEE Workshop on Signal Processing Systems*, 2014.
[26] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[27] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE TPAMI*, 35(1):221–231, 2013.

[28] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*, 2016.

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

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

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

[32] Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[33] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *CVPR*, 2015.

[34] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, pages 1–23, 2016.

[35] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *ICML*, 2009.

[36] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[37] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[38] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *IEEE TPAMI*, 2016.

[39] O. Rippel, J. Snoek, and R. P. Adams. Spectral representations for convolutional neural networks. In *NIPS*, 2015.

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[42] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *NIPS*, 2014.

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

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

[45] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[46] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[47] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, 2015.

[48] G. K. Wallace. The jpeg still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1):xviii–xxxiv, 1992.

[49] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[50] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu. Cnnpack: Packing convolutional neural networks in the frequency domain. In *NIPS*, 2016.

[51] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. Hcp: A flexible cnn framework for multi-label image classification. *IEEE TPAMI*, 38(9):1901–1907, 2016.

[52] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

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

[54] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *NIPS*, 2014.

[55] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Yunhe Wang received the B.E degree from Xidian University in 2013. Currently, he is a Ph.D. candidate with the Key Laboratory of Machine Perception (Ministry of Education) in the Peking University. His research interests lie primarily in machine learning and computer vision.

Chang Xu received the B.E. degree from Tianjin University, China, and the Ph.D. degree from Peking University, China. He is currently a Lecturer with the School of Information Technologies and the Faculty of Engineering and Information Technologies in the University of Sydney. His research interests lie primarily in machine learning, multimedia search and computer vision.

Chao Xu received the B.E. degree from Tsinghua University in 1988, the M.S. degree from University of Science and Technology of China in 1991 and the Ph.D degree from Institute of Electronics, Chinese Academy of Sciences in 1997. Between 1991 and 1994 he was employed as an assistant professor by University of Science and Technology of China. Since 1997 Dr. Xu has been with School of EECS at Peking University where he is currently a Professor. His research interests are in image and video coding, processing and understanding. He has authored or co-authored more than 80 publications and 5 patents in these fields.

Dacheng Tao (F'15) is Professor of Computer Science and ARC Future Fellow in the School of Information Technologies and the Faculty of Engineering and Information Technologies, and the Founding Director of the UBTech Sydney Artificial Intelligence Centre at the University of Sydney. He was Professor of Computer Science and Director of Centre for Artificial Intelligence in the University of Technology Sydney. He mainly applies statistics and mathematics to Artificial Intelligence and Data Science. His research interests spread across computer vision, data science, image processing, machine learning, and video surveillance. His research results have expounded in one monograph and 500+ publications at prestigious journals and prominent conferences, such as IEEE T-PAMI, T-NNLS, T-IP, JMLR, IJCV, NIPS, CIKM, ICML, CVPR, ICCV, ECCV, AISTATS, ICDM; and ACM SIGKDD, with several best paper awards, such as the best theory/algorithm paper runner up award in IEEE ICDM'07, the best student paper award in IEEE ICDM'13, and the 2014 ICDM 10-year highest-impact paper award. He received the 2015 Australian Scopus-Eureka Prize, the 2015 ACS Gold Disruptor Award and the 2015 UTS Vice-Chancellor's Medal for Exceptional Research. He is a Fellow of the IEEE, OSA, IAPR and SPIE.