Iterative Deep Model Compression and Acceleration in the Frequency Domain

Yao Zeng ZENGYAO@SMAIL.NJU.EDU.CN
Nanjing University

Xusheng Liu XUSHENGLIU_SGCSC@163.COM
State Grid Customer Service Center

Lintan Sun LINTANSUN@163.COM
State Grid Customer Service Center

Wenzhong Li LWZ@NJU.EDU.CN
Nanjing University

Yuchu Fang FANGYUCHU@SMAIL.NJU.EDU.CN
Nanjing University

Qingning Lu QNLU@SMAIL.NJU.EDU.CN
Nanjing University

Sanglu Lu SANGLU@NJU.EDU.CN
Nanjing University

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Deep Convolutional Neural Networks (CNNs) are successfully applied in many complex tasks, but their storage and huge computational costs hinder their deployment on edge devices. CNN model compression techniques have been widely studied in the past five years, most of which are conducted in the spatial domain. Inspired by the sparsity and low-rank properties of weight matrices in the frequency domain, we propose a novel frequency pruning framework for model compression and acceleration while maintaining high-performance. We firstly apply Discrete Cosine Transform (DCT) on convolutional kernels and train them in the frequency domain to get sparse representations. Then we propose an iterative model compression method to decompose the frequency matrices with a sample-based low-rank approximation algorithm, and then fine-tune and recompose the low-rank matrices gradually until a predefined compression ratio is reached. We further demonstrate that model inference can be conducted with the decomposed frequency matrices, where model parameters and inference cost can be significantly reduced. Extensive experiments using well-known CNN models based on three open datasets show that the proposed method outperforms the state-of-the-arts in reduction of both the number of parameters and floating-point operations (FLOPs) without sacrificing too much model accuracy.

Keywords: Compression, DCT, Low-rank Approximation.

1. Introduction

Convolutional neural networks (CNNs) have been applied in many complex tasks such as image classification and object detection because of their capability of representation learning.

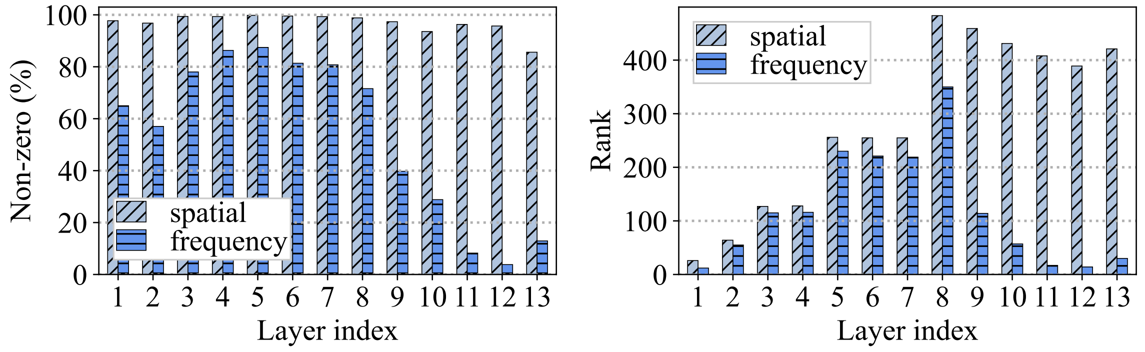


Figure 1: Comparison of sparsity (left) and rank (right) of different layers of VGG16 on CIFAR10.

However, with the CNNs becoming deeper and deeper, they should occupy much memory storage and computational cost, which prevents them from being deployed in resource-limited devices. Model compression, which focuses on reducing the size of CNNs and accelerating their inference or training time, has drawn much attention from academia and industry.

Neural network pruning Han et al. (2015); Li et al. (2017); Liu et al. (2017) has been widely studied in the past few years, and most of them are conducted in the spatial domain. Unstructured pruning Frankle and Carbin (2019); Han et al. (2015); Lee et al. (2019) reduces the number of non-zero parameters by inducing sparsity to weight matrices. Although it can obtain a large compression rate on model size, the speeding up demands for specific hardware/libraries support. Structured pruning Li et al. (2017); Liu et al. (2017) achieves practical acceleration, but the instability of its quality prevents it from moving towards higher pruning rate.

Recent works Liu et al. (2018); Wang et al. (2018); Xu et al. (2020) showed that due to the nature of local smoothness of images, filters in CNNs tend to be smooth, and they are dominated by low-frequency components in the frequency domain. Using frequency domain to compress CNNs was proposed recently. *CNNpack* Wang et al. (2018) compressed CNN models in the frequency domain by discarding a large number of low-energy frequency coefficients so as to reduce storage. *FreshNets* Chen et al. (2016) employed a hashing trick for random weight sharing and compressed parameters in a frequency-sensitive fashion such that low-frequency components are better preserved. *FDNP* Liu et al. (2018) dynamically pruned frequency-domain coefficients to reduce model parameters based on different importance of frequency bands in the weight matrix. Despite the efforts being made, they do not fully capture the potentials for model compression in the frequency domain. Figure 1 compares the percentage of non-zero entries and the rank of weight matrices in different layers of VGG16 model trained on CIFAR10 dataset Krizhevsky (2009) in both spatial and frequency domain. It is shown that the weight matrices in the frequency domain are much sparser and have lower ranks, which can be exploited to develop efficient compression algorithms for CNN models. To leverage such properties, we propose a novel frequency model compression method called *FreqPrune* to reduce model parameters and accelerate inference without causing significant performance loss. We transform the convolutional layers as well as their input featuremaps into the frequency domain using Discrete Cosine Transform (DCT) and show the feasibility of replacing convolutional operation by frequency matrix multiplication. Then we sparsify the frequency matrices of a CNN model by retraining them with a limited number of epochs using a sparsity regularization. Base on that, we propose an iterative model compression method, which decomposes the frequency matrices with a sample-based low-rank approximation algorithm to obtain compact model representation and find-tune the model iteratively until a predefined compression ratio is reached. We demonstrate that model retraining and inference can be conducted with the decomposed frequency matrices, based on which the model parameters can be compressed, and the inference cost measured by FLOPs (floating-point operations) can be significantly reduced.

Compared to the conventional deep model compression methods, the advantages of the proposed FreqPrune method are as follows. *(1) Adaptiveness:* unlike conventional filter pruning methods requiring manual setting of pruning rate for each layer, the proposed method can automatically and adaptively decide the pruning rate without expert knowledge. *(2) Effectiveness:* with the sparsification and low-rank decomposition method introduced in the frequency domain, the proposed method can reduce both FLOPs and the number of model parameters significantly and efficiently.

The contributions of the paper are summarized as follows:

- We propose a novel iterative deep model compression and acceleration method in the frequency domain. We transform spatial convolutional operation as frequency matrix multiplication, and then sparsify and decompose the frequency matrices using low-rank approximation and fine-tuning techniques iteratively to obtain a compact model representation. We show that model inference can be conducted with the decomposed frequency matrices, which can reduce both computational and storage costs.

- To the best of our knowledge, we are the first to comprehensively explore model decomposition, weight pruning, parameter fine-tuning, and inference in the frequency domain. Unlike the previous work of weight pruning in the frequency domain, our work obtains practical model compression and acceleration, reducing more FLOPs by low-rank decomposition and inference in the frequency domain.

- We explore the importance of components in the frequency domain that can easily distinguish salient components from trivial ones. We search for important components efficiently without trial-and-error or reinforcement learning, which are time-consuming.

- We conduct extensive experiments on a variety of CNN models based on three open datasets. It shows that the proposed method outperforms the state-of-the-art model compression methods to reduce FLOPs and the number of parameters without harming the test accuracy.

2. Related Works

We summarize the related works in the following aspects: unstructured model pruning, structured model pruning, and model compression in the frequency domain.

Unstructured model pruning aims to remove useless parameters from CNNs and turn the dense weight matrices into sparse ones. LeCun et al. (1990) tried to prune the unimportant weights using the second-order Taylor expansion. Han et al. (2015) proposed an iterative weight pruning method by discarding weights with small absolute values. Lee et al. (2019) proposed to prune the network at initialization by identifying structurally important connections. The lottery ticket hypothesis Frankle and Carbin (2019) showed that dense networks contain sparse sub-networks that have comparable performance to the original networks. However, unstructured pruning relied on dedicate hardware for inference acceleration Li et al. (2017).

Structured model pruning adopted a filter-wise pruning mechanism to simplify the CNN structure and accelerate inference. Many works utilized the discriminative properties of a pre-trained CNN model to measure the importance of its filters, such as l_p norm He et al. (2018a); Li et al. (2017), scaling factor Liu et al. (2017), correlation Wang et al. (2019a), and reconstruction error Lin et al. (2020a). Recently, some AutoML methods were introduced to prune networks automatically. He et al. (2018b) and Lin et al. (2017) utilized reinforcement learning to decide the pruning ratio of each convolutional layer. Based on the finding of Liu et al. (2019) that a pruned network could be trained from scratch to reach comparable accuracy against the full model, Wang et al. (2019b) proposed an algorithm to prune networks in the very early stage.

Model compression in the frequency domain was proposed recently. Xu et al. (2020) proposed a learning-based frequency selection method to identify the trivial frequency components in images which can be removed without accuracy loss. Wang et al. (2018) compressed CNNs in the frequency domain by decomposing filter representation into common parts and individual private parts so that a large number of low-energy frequency coefficients in both parts can be discarded. Chen et al. (2016) converted filter weights to the frequency domain and used hash functions to group them into hash buckets to save storage. Liu et al. (2018) dynamically pruned frequency-domain coefficients utilizing the different importance of frequency bands.

3. FreqPrune Model Compression Method

In this paper, we propose a novel pruning framework named *FreqPrune* based on iterative weight matrices decomposition and recomposition in the frequency domain.

3.1. Framework Overview

Figure 2 shows the pipeline of FreqPrune. Firstly, we transform the convolutional layers as well as its input featuremaps into frequency domain with Discrete Cosine Transform (DCT) Rao et al. (1990) to replace convolutional operation by frequency matrix multiplication (Section 3.2.1). Secondly, we derive the gradient for backward propagation in the frequency domain to retrain the deep model. We sparsify the weight matrix by regularization to concentrate the information into some certain frequency regions without accuracy drop (Section 3.2.2). Thirdly, we propose a sample-based low-rank approximation algorithm to decompose the frequency weight matrices (Section 3.3.1), and fine-tune the decomposed matrices by backward propagation to recover the test accuracy (Section 3.3.2). Finally, the

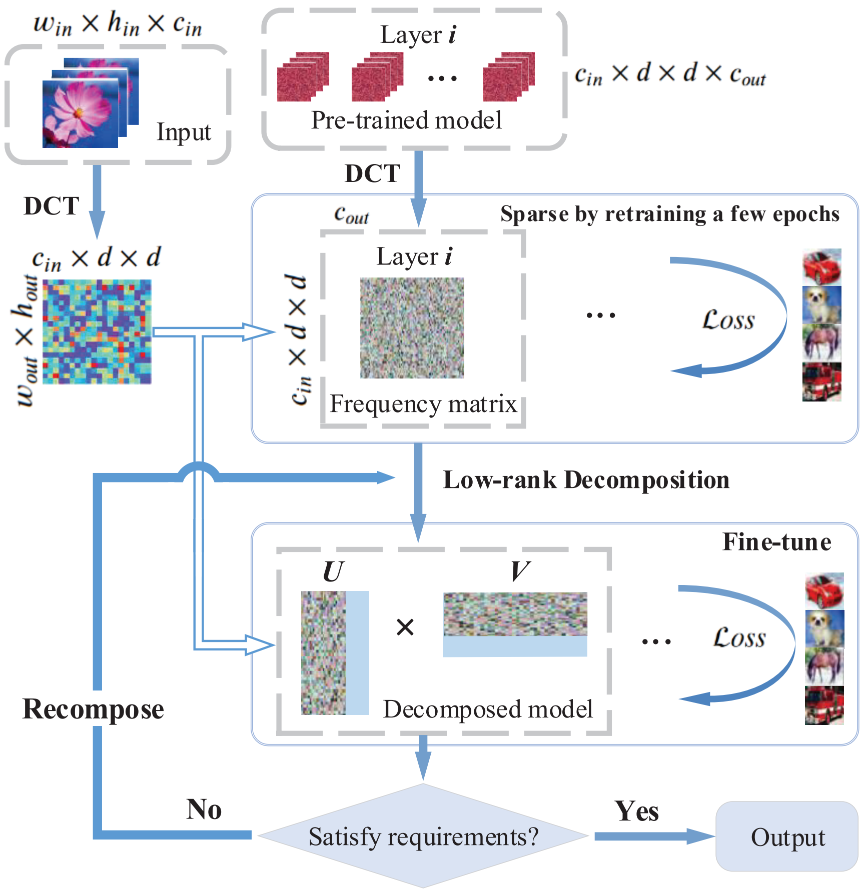


Figure 2: Pipeline of model compression in the frequency domain. We firstly transform the convolutional layers into frequency domain with DCT. Then, we sparsify the weight matrix to concentrate on the information in some certain frequency regions. Finally, we decompose the frequency weight matrices, fine-tune the decomposed matrices, and recompose them for further rank reduction until the required compression ratio is satisfied.

decomposed matrices are recomposed for further rank reduction until the required compression ratio is satisfied (Section 3.3.3).

3.2. Model Sparsification in the Frequency Domain

Given a pretrained deep model, we first apply DCT to transform the convolutional layers into frequency representation, and then retrain the model with sparse regularization.

3.2.1. DCT for CNNs

We introduced the principle of DCT to transform a CNN model into the frequency domain. Denote the DCT as $\mathbf{D}$. Given an input matrix $\mathcal{P} \in \mathbb{R}^{n\times n}$, the DCT coefficient $\mathcal{F} \in \mathbb{R}^{n\times n}$ of $\mathcal{P}$ is defined as: $\mathcal{F} = \mathbf{D}(\mathcal{P})$. In element-wise, let $\mathcal{P} = [\mathcal{P}_{ij}]_{n\times n}$ and $\mathcal{F} = [\mathcal{F}_{uv}]_{n\times n}$, the DCT can be denoted as:

$$\mathcal{F}_{uv} = \alpha_u \alpha_v \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \mathcal{P}_{ij} \cos \frac{\pi(2i+1)u}{2n} \cos \frac{\pi(2j+1)v}{2n}, \tag{1}$$

where $\alpha_u = \sqrt{\frac{1}{n}}$ if $u = 0$ and $\alpha_u = \sqrt{\frac{2}{n}}$, otherwise.

Once the transformation pairs both satisfy dividable and symmetry, which means that the patch $\mathcal{P}$ is square, the matrix form of DCT can be written as $\mathcal{F} = C\mathcal{P}C^T$, where $C = [C_{ij}]_{n\times n}$ is the transformation matrix, and the element of C is $C_{ij} = \alpha_i \cdot \cos \frac{\pi(2i+1)j}{2n}$.

Assume a convolutional layer consists of c_{out} filters, each of them has c_{in} kernels sized $d \times d$. Denote a convolutional layer as a 4D tensor $\widetilde{W} \in \mathbb{R}^{c_{in}\times d\times d\times c_{out}}$, and the input as a 3D tensor $\widetilde{I} \in \mathbb{R}^{c_{in}\times h_{in}\times w_{in}}$, where $h_{in} \times w_{in}$ is the dimension of the input feature map. Denote the output of convolution as $\widetilde{O} \in \mathbb{R}^{c_{out}\times h_{out}\times w_{out}}$, where $h_{out} \times w_{out}$ is the dimension of the output feature map. The convolution operation (in the spatial domain) can be denoted by $mat(\widetilde{O}) = mat(\widetilde{I}) \cdot mat(\widetilde{W})$, where $mat(\cdot)$ is the matrix form of a tensor[1].

Let W, I and O be the frequency representations of $\widetilde{W}$, $\widetilde{I}$ and $\widetilde{O}$ accordingly. They are formed as follows. For a convolutional layer, we apply DCT on each kernel $d \times d$ using Eq. (1) to flatten it into a vector, and group all vectors in the layer to form the frequency weight matrix $W \in \mathbb{R}^{(d\times d\times c_{in})\times c_{out}}$. Similarly, we apply DCT to the input tensor to form its frequency representation $I \in \mathbb{R}^{(h_{out}\times w_{out})\times(c_{in}\times d\times d)}$.

Next we illustrate that convolution operation in the spatial domain can be transformed to matrix multiplication in the frequency domain. The DCT of $\widetilde{W}$ is $W = \mathbf{D}(\widetilde{W}) = D \cdot mat(\widetilde{W})$, where D is an orthogonal matrix defined by Wang et al. (2018). Similarly, the DCT of $\widetilde{I}$ is $I = \mathbf{D}(\widetilde{I}) = mat(\widetilde{I}) \cdot D^T$. Therefore we have

$$O = I \cdot W = \left[mat(\widetilde{I}) \cdot D^T \right] \cdot \left[D \cdot mat(\widetilde{W}) \right] = mat(\widetilde{O}). \tag{2}$$

With such property, we can represent the original CNN model as a sequence of matrices in the frequency domain.

3.2.2. Retraining for weights sparsification

After frequency transformation, we retrain the model for a few epochs to sparsify its weight representation. The retraining process is conducted in the frequency domain as follows. In forward propagation, the output of each layer is simply computed by Eq. (2). In back propagation, the gradient used to update W can be calculated by

$$\frac{\partial \mathcal{L}}{\partial W_{mn}} = \frac{\partial \mathcal{L}}{\partial O_{ij}} \left(J^{mn} I^T \right)_{ij}, \tag{3}$$

where $\mathcal{L}$ is the objective function, $(\cdot)_{ij}$ means the element of the matrix in position (i, j), and J^{mn} is a single-entry matrix where the (m, n)-th element is 1 and 0 elsewhere.

To get a sparse representation in the frequency domain, we add a regularization term to force more weights to be zero as much as possible. Therefore we use the following objective function to retrain the model.

$$\mathcal{L}_{total} = \mathcal{L}_{original} \left(\mathbf{D}(model) \right) + \lambda \| \mathbf{D}(model) \|_1, \tag{4}$$

1. https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html

where $\mathcal{L}_{original}()$ is the loss function of the original deep model, $\mathbf{D}(model)$ is the model's weights (after DCT) in the frequency domain, $\|\cdot\|_1$ is the l_1-norm regularization that drives the weights approaching zero, and λ is a weight to balance the degree of sparsity.

3.3. Iterative Model Compression & Acceleration

We propose an iterative decomposition-recomposition method for model compression and acceleration as follows.

3.3.1. Sample-based Low-rank Approximation

To reduce the parameter size and computational cost of the CNN model (after DCT), we introduce a sample-based low-rank approximation method to decompose the weight matrices into multiplication of smaller matrices while producing similar output. Given a weight matrix W, low-rank approximation seeks two matrices U and V to decompose W, i.e., $W \approx UV$, where $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{r\times n}$, and $r \leq \min\{m, n\}$. A natural choice for the approximation criterion is to minimize $\|W - UV\|_F^2$ where $\|\cdot\|_F$ is the Frobenius norm Horn and Johnson (2012).However, an optimal rank for matrix decomposition is NP-hard Zhou and Tao (2013), and a layer-wise matrix decomposition approach neglects the information flow through CNN layers, which could cause error accumulation. To address these issues, we propose a low-rank approximation method based on small samples.

Consider a set of samples $\Omega = \{\omega_1, \omega_2, \cdots\}$, which is a subset of the input dataset. For a sample ω_i, it goes through several layers of convolutional operations, where the j-th layer's output $\omega_i^{(j)}$ is used as the input of the $(j+1)$-th layer. Without causing confusion, we omit the superscript thereafter, and generally refer to the layer's input as ω_i. Based on the sample set, the low-rank approximation problem can be formulated as

$$\min_{U,V} \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} \|\omega_i W - \omega_i UV\|_F^2, \tag{5}$$

$$\text{s.t.} \quad \frac{\|W - UV\|_F^2}{\|W\|_F^2} \leq \varepsilon, \tag{6}$$

$$rank(UV) \leq rank(W) - 1, \tag{7}$$

where $\varepsilon \in (0, 1)$ is a predefined error tolerance and the proposed low-rank approximation can be applied on any W (full rank or not) to minimize Eq. (5).

The above optimization problem seeks a UV decomposition of the weight matrix W with two constraints: (1) the difference between the decomposed matrix and the original one is within a predefined threshold (Eq. (6)), and (2) the rank of the decomposed matrix is lower than that of the original matrix (Eq. (7)). By fixing the rank of UV to $rank(W) - 1$, we can derive a feasible solution of the optimization problem as follows. By introducing a Lagrange multiplier γ, the Lagrange function with respect to U, V, γ can be written as

$$\mathcal{L}_{U,V,\gamma} = \frac{1}{2|\Omega|} \sum_{i=1}^{|\Omega|} \|\omega_i W - \omega_i UV\|_F^2 + \frac{\gamma}{2}\left(\|W - UV\|_F^2 - \varepsilon \|W\|_F^2\right).$$

Taking partial derivative on $\mathcal{L}$, we have

$$\frac{\partial \mathcal{L}}{\partial U} = \left(-\frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|}\omega_i^T\omega(W-UV)-\gamma(W-UV)\right)V^T,$$
$$\frac{\partial \mathcal{L}}{\partial V} = U^T\left(-\frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|}\omega_i^T\omega(W-UV)-\gamma(W-UV)\right).$$

Let $\frac{\partial \mathcal{L}}{\partial U}=0$ and $\frac{\partial \mathcal{L}}{\partial V}=0$, we use ADMM (Alternating Direction Method of Multipliers) to alternatively update U and V in step k as follows,

$$\begin{aligned} U_k &= A^\dagger B V_{k-1}^T (V_{k-1}V_{k-1}^T)^\dagger, \\ V_k &= (U_k^T A U_k)^\dagger U_k^T B, \end{aligned} \tag{8}$$

where $A=\frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|}\omega_i^T\omega_i+\gamma E$; $B=\frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|}\omega_i^T\omega_i W+\gamma W$; † means the MP inverse Penrose (1955) of the matrix, and E is an unit matrix.

We update U_k and V_k iteratively until convergence, and if the constraint of Eq. (6) holds, the matrices U and V are output as the decomposition of W.

Based on the above low-rank approximation algorithm, we can sequentially decompose the CNN model in the frequency domain layer-by-layer using the sample sets Ω. After optimizing the current layer, we use its output as the input sample of the next layer and decompose the next layer in the same way without causing error accumulation through layers.

The pseudo-code of the algorithm is illustrated in Algorithm 1.

3.3.2. Fine-tuning

After low-rank decomposition, the CNN model can be represented by a sequence of low-rank matrices U and V. Like conventional filter pruning approaches He et al. (2018a); Li et al. (2017), we use the original dataset to fine-tune the compressed model to recover its' accuracy.

Different from the existing works, we fine-tune the decomposed low-rank matrices U and V in the frequency domain. To use the decomposed low-rank matrices for model training and inference, we explicitly derive the forward and backward propagation process.

For each layer, the forward propagation is simply computed by

$$I \cdot U \cdot V = O. \tag{9}$$

To illustrate the backward propagation process, we derive the gradients of different components U and V for a given loss function $\mathcal{L}$.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial V_{mn}} &= \frac{\partial \mathcal{L}}{\partial O_{ij}^T}\cdot\frac{\partial O_{ij}^T}{\partial V_{mn}} = \frac{\partial \mathcal{L}}{\partial O_{ij}^T}(J^{nm}U^T I^T)_{ij}, \\ \frac{\partial \mathcal{L}}{\partial U_{mn}} &= \frac{\partial \mathcal{L}}{\partial O_{ij}}\cdot\frac{\partial O_{ij}}{\partial O'_{kl}}\cdot\frac{\partial O'_{kl}}{\partial U_{mn}} \quad (O'=U^T I^T) \\ &= \frac{\partial \mathcal{L}}{\partial O_{ij}}\cdot(J^{lk}V)_{ij}\cdot(J^{nm}I^T)_{kl}, \end{aligned} \tag{10}$$

Algorithm 1 Iterative Rank-reduction in Frequency-domain

```
Input: $model$:original network; $\varepsilon_0$:initialized error tolerance; $\Delta\varepsilon$:increment of error tolerance; $\lambda$:regular coefficient to control sparsity; $\mathcal{T}$:max rounds; FLOPs requirements;
Output: decomposed compact model;
1: Module 1: Sparsing in the frequency domain
2: Transform $\mathcal{W}$ into $W$ for all layers to get $\mathbf{D}(model)$;
3: repeat
       Feed data to $\mathbf{D}(model)$ to calculate loss Eq. (4);
       Compute gradient by Eq. (3) to fine-tune;
   until Performance of $\mathbf{D}(model)$ converged.;
4: Module 2: Iterative Rank Reduction
5: $round = 0$;
6: while $round \leq \mathcal{T}$ do
       foreach layers in $\mathcal{D}(model)$ do
           if $rank(W) - 1 > 0$ then
               Initialize $U_0V_0$ with rank equal to $rank(W)-1$; for $k = 1$ to $K$ do
                   Update $U_k$ and $V_k$ by Eq(8);
               end
               if Eq(6) satisfied then
                   decompose $W$ into $U_kV_k$;
               end
           end
       end
       Fine-tune $U, V$ for all layers;
       if FLOPs or #Params satisfied then
           return decomposed model;
       else
           Recompose $UV$ to generate $W'$; $W :\leftarrow W'$
       end
   end
```

where the gradients $\frac{\partial \mathcal{L}}{\partial O_{ij}^T}$ and $\frac{\partial \mathcal{L}}{\partial O_{ij}}$ are given by Eq. (3). With the above gradients, U and V can be updated using the stochastic gradient descent method in the forward propagation process.

3.3.3. Recomposition

After fine-tuning, we recompose the low-rank matrices U and V into a new weight matrix $W'{=}UV$. Since $\text{rank}(W') \leq \text{rank}(W)$, model inference based on the low rank matrices W' yields lower computational cost.

Model	Method	Frr(%)	Prr(%)	Acc.(%)	Baseline(%)	ΔAcc.(%)
VGG-16	Li-pruning	34.2	64	93.4	93.66	-0.26
	SFP*	73.41	73.13	93.18	93.69	-0.51
	COP	73.5	92.8	93.31	93.66	-0.35
	FDNP*	70.89	75.37	92.76	93.69	-0.93
	FreqPrune-A	71.65	80.46	**93.51**	93.69	**-0.18**
	FreqPrune-B	**75.94**	**93.28**	93.32	93.69	-0.37
ResNet-56	Li-pruning	27.6	13.7	93.06	93.04	**+0.02**
	SFP	52.6	-	93.35	93.59	-0.24
	GAL	60.2	65.9	91.58	93.26	-1.68
	ABCPruner	54.13	54.2	93.23	93.26	-0.03
	EB-Tickets*	60	59.21	92.74	93.5	-0.76
	FDNP*	58.97	55.03	93.07	93.50	-0.43
	FreqPrune-A	60.08	54.95	**93.37**	93.5	-0.13
	FreqPrune-B	**62.67**	**72.46**	93.23	93.5	-0.27
ResNet-110	Li-pruning	38.6	32.4	93.3	93.53	-0.23
	SFP	40.8	-	93.38	93.68	-0.3
	GAL	48.5	44.8	92.74	93.5	-0.76
	FDNP	-	75	93.5	-	-
	FreqPrune-A	41.23	55.49	**94.08**	94.02	**+0.06**
	FreqPrune-B	**63.58**	**81.29**	93.24	94.02	-0.78

Table 1: Results on CIFAR10. Results with "*" are reproduced from their released code, and the others are from the original papers. "Frr(%)" means the FLOPs-reductoin ratio, and "Prr(%)" means the parameters-reduction ratio. "Acc(%)" means top-1 test accuracy.

3.3.4. Iterative Model Compression Algorithm

To avoid unrecoverable accuracy loss caused by aggressive parameter pruning, we adopt an iterative decomposition-recomposition method for gradually model compression in the frequency domain. It contains the following steps.

(1) Given a deep CNN model, we use the frequency transformation method introduced in Section 3.2.1 to represent it by a sequence of frequency weight matrices.

(2) For each weight matrix W, we choose a small sample set Ω, and apply the sample-based low-rank approximation algorithm introduced in Section 3.3.1 to decompose it into low rank matrices U and V with error tolerance ε. If there is no feasible decomposition for W, it skips and tries to decompose the next weight matrix.

(3) To recover the model accuracy, we fine-tune the decomposed low-rank matrices U and V using the method introduced in Section 3.3.2.

(4) After fine-tuning, we recompose $W' = UV$, where W' has lower rank than W and fewer computational cost (# FLOPs) in inference.

(5) Let $W = W'$. Repeat step (2)-(4), until the required compression ratio (measured by # FLOPs) is satisfied, or the maximum number of iteration rounds is reached.

4. Experiments

In this section, we conduct extensive experiments on a variety of deep CNN models based on several datasets to evaluate the performance of the proposed FreqPrune method for model compression and acceleration.

4.1. Experimental Setup

Implement Details: We implement FreqPrune[2] using PyTorch v1.3. The experiments are conducted on a GPU-equipped server (Tesla V-100). We train the baseline networks following the same training settings as Li et al. (2017); Liu et al. (2019). After transforming the networks with DCT, we sparsify the models by training them with 20 epochs on CIFAR and 5 epochs on ImageNet using Eq. (4). We set the learning rate to 0.001 and 0.0001 for CIFAR and ImageNet respectively. The frequency matrices are iteratively decomposed with a given error tolerance ratio ε, and the models are fine-tuned 50 epochs for CIFAR and 30 epochs for ImageNet. We adopt SGD as the optimizer for all training processes with weight decay being 10^{-4} and momentum being 0.9. The default size of sample set Ω is 5% of the total dataset, and the default error tolerance rate is $\varepsilon = 0.02$.

Models and Datasets: We conduct experiments based on two well-known CNN models: VGG Krizhevsky (2009) and ResNet He et al. (2016). We use three open datasets: CIFAR10 Krizhevsky (2009), CIFAR100 Krizhevsky (2009) and ImageNet Deng et al. (2009). We implement two versions of FreqPrune: (1) *FreqPrune-A*: It prunes the same FLOPs as the state-of-the-arts to show the accuracy achieved; (2) *FreqPrune-B*: It keeps the same accuracy as the state-of-the-arts to show the compression ratio measured by the reduction of FLOPs and the number of parameters.

Baseline Algorithms: We compare FreqPrune with the following state-of-the-art model compression methods. (1) *Unstructured pruning*: Rewind Renda et al. (2020). (2) *Structured pruning*: Li-pruning Li et al. (2017), SFP He et al. (2018a), COP Wang et al. (2019a), GAL Lin et al. (2019), ABCPruner Lin et al. (2020b), PFA Suau et al. (2018), CP He et al. (2017), EB-Tickets You et al. (2020), FilterSketch Lin et al. (2020a) and PP Singh et al. (2019). (3) *Frequency pruning*: FDNP Liu et al. (2018).

4.2. Results on CIFAR

The results of model compression on the CIFAR10 and CIFAR100 datasets are shown in Table 1 and 2.

For the results on CIFAR10 (illustrated in Table 1), FreqPrune-A achieves the highest test accuracy while keeping the same pruned FLOPs as that of the other algorithms. FreqPrune-B can reduce much more FLOPs and parameters than other algorithms but still acquires comparable accuracy. The results of FreqPrune-A in compressing ResNet110 is especially stunning, which reduce 41.23% FLOPs and 55.49% parameters while the accuracy of the compressed model achieves 94.08%, which is even higher than the baseline accuracy by 0.06%.

Similar results can be found from the experiments on CIFAR100 (Table 2). FreqPrune reduces more FLOPs and parameters from VGG and ResNet than other algorithms with

2. Source code is available at https://github.com/AldrichZeng/FreqPrune_ACML2021

Model	Method	Frr(%)	Prr(%)	Acc.(%)	Baseline(%)	ΔAcc.(%)
VGG-16	SFP*	42.28	42.35	71.53	72.59	-1.06
	COP	43.1	73.2	71.77	72.59	-0.82
	PFA-KL	46.1	58.1	70	72.59	-2.59
	EB-Tickets*	44.96	78.16	67.92	72.58	-4.66
	FDNP*	50.68	60.31	68.93	72.58	-3.65
	FreqPrune-A	52.96	66.13	**71.9**	72.58	**-0.68**
	FreqPrune-B	**66.51**	**78.55**	71.7	72.58	-0.88
ResNet-32	SFP*	33.9	33.8	67.83	70.01	-2.18
	COP	34.2	35.2	68.29	68.74	-0.45
	EB-Tickets*	41.94	21.83	68.03	70.01	-1.98
	FDNP*	42.60	46.37	68.94	70.01	-1.07
	FreqPrune-A	42.56	47.17	**69.73**	70.01	**-0.28**
	FreqPrune-B	**47.98**	**53.95**	68.87	70.01	-1.14
ResNet-56	SFP*	54.06	40.99	69.38	71.33	-1.95
	PFA-KL	32.3	26.4	68.05	-	-
	PFA-En	20.6	18.5	69.22	-	-
	EB-Tickets*	53.88	26.14	68.68	71.33	-2.65
	FDNP*	53.08	49.56	68.74	71.33	-2.59
	FreqPrune-A	55.43	47.59	**70.46**	71.33	**-0.87**
	FreqPrune-B	**62.43**	**59.85**	70.09	71.33	-1.24

Table 2: Results on CIFAR100.

very low accuracy drop. As an example, FreqPrune-B outperforms other methods by pruning dramatically more parameters (nearly 20% more) from ResNet56 and achieves the highest accuracy.

4.3. Results on ImageNet

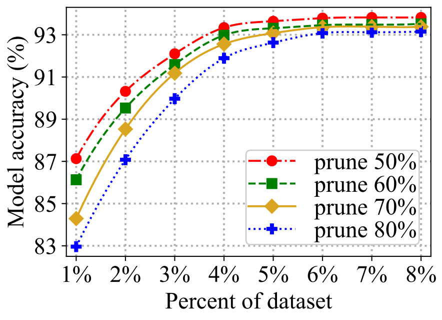
We further conduct extensive experiments on a real-world large-scale dataset: ImageNet. We compress ResNet34 and ResNet56 using FreqPrune and compare the results with the state-of-the-art methods.

Due to the existence of 1×1 kernels in ResNet on ImageNet, the frequency coefficients in the frequency domain equal to the weights in the spatial domain. Thus, we can skip DCT transformation on the convolutional layers whose kernels are sized 1×1. Nevertheless, we still conduct low-rank approximation on the weight matrices to compress these layers.

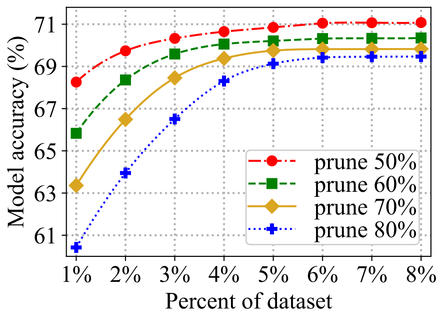
As demonstrated in Table 3, FreqPrune-A can reduce 42.02% FLOPs and 43.92% parameters from ResNet34, both of which are higher than the state-of-the-art methods, and the accuracy of the compressed network is still the best with Top-1 accuracy reaching 72.32% and surpassing other methods by a great margin. For ResNet34, FreqPrune-B can significantly reduce FLOPs (up to $3\times$ pruned FLOPs) and parameters with minor accuracy drop. For ResNet56, FreqPrune-A gets the second highest Top-1 accuracy 74.59% and the highest Top-5 accuracy 91.03% with lowest accuracy drop 1.54% while pruning 20% more FLOPs than that of SFP He et al. (2018a). In the meanwhile, FreqPrune-B outperforms other methods in terms of Top-1 accuracy when pruning the same level of FLOPs. These validates the effectiveness of FreqPrune on real-world large-scale dataset.

Model	Method	Frr(%)	Prr(%)	Acc.(%)		Baseline(%)		ΔAcc.(%)	
				Top1	Top5	Top1	Top5	Top1	Top5
ResNet-34	Li-pruning	24.2	10.8	72.17	-	73.23	-	-1.06	-
	SFP	41.1	-	71.83	90.33	73.42	91.36	-1.59	-1.03
	Rewinding	20.63	-	72.1	-	73.6	-	-1.5	-
	FDNP*	40.35	47.18	71.76	90.07	73.27	91.43	-1.51	-1.36
	FreqPrune-A	42.02	43.92	**72.32**	**91.03**	73.27	91.43	**-0.95**	**-0.4**
	FreqPrune-B	**58.73**	**59.48**	71.79	90.28	73.27	91.43	-1.48	-1.15
ResNet-50	SFP	41.8	-	**74.61**	92.06	76.15	92.87	**-1.54**	-0.81
	CP	51.22	-	73.30	-	76.10	-	-2.8	-
	PP	52.2	46.27	-	91.4	-	92.2	-	-0.8
	FilterSketch	63.1	59.2	73.04	91.18	76.13	92.86	-3.09	-1.68
	GAL	72.86	59.96	69.31	89.12	76.15	92.87	-6.84	-3.75
	FDNP*	59.73	60.93	73.2	91.38	76.13	92.86	-2.93	-1.48
	FreqPrune-A	61.06	57.81	74.59	**92.15**	76.13	92.86	**-1.54**	**-0.71**
	FreqPrune-B	**67.85**	**62.09**	73.05	91.16	76.13	92.86	-3.08	-1.70

Table 3: Results on ImageNet.



(a) VGG16 on CIFAR10



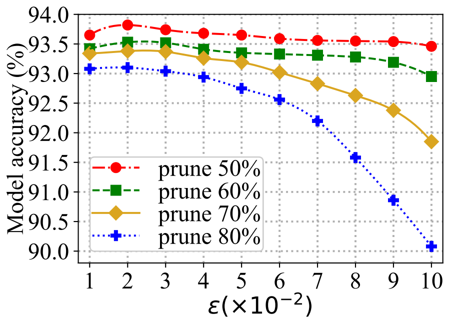
(b) ResNet56 on CIFAR100

Figure 3: Influence of the size of sample set Ω.

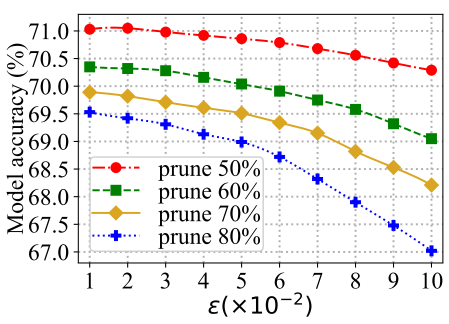
4.4. Hyperparameter Sensitivity

4.4.1. Influence of size of sample set Ω

In the optimization problem of Eq. (5), the size of sample set Ω will influence the result of low-rank approximation. We explore the model compression performance by randomly choosing different percentage of samples from the original dataset to form Ω. Figure 3 shows the model accuracy of VGG16 and ResNet56 on different datasets with increasing percentage of sample size. We make the following observations from the figure. (1) The model accuracy is relatively low under few samples. A possible reason is that matrix decomposition of UV in Eq. (5) based on fewer samples are more likely to be under-fitting. (2) As the number of samples increases, the decomposed UV tends to be stable, and the accuracy of the fine-tuned model is improved. (3) Sampling more than 5% of the datasets brings marginal benefit to the model accuracy, therefore randomly sampling 5% of the datasets for low-rank decomposition is enough to achieve high performance.



(a) VGG16 on CIFAR10



(b) ResNet56 on CIFAR100

Figure 4: Influence of the error tolerance rate ε.

4.4.2. Influence of error tolerance rate ε

In the optimization process of Eq. (5)(6)(7), the error tolerance ε determines the granularity of frequency pruning, where a larger ε results in a lower-rank decomposition. We explore the model accuracy of VGG16 and ResNet56 with increasing ε, and the results are illustrated in Figure 4. We make the following observations from the figure. (1) For $\varepsilon \leq 0.05$, the influence to model accuracy is negligible. The reason is that for small ε, the information loss in matrix decomposition is minor, and the model accuracy can easily be restored after fine-tuning. (2) For larger ε, the model accuracy drops rapidly. The reason is that larger ε leads to more aggressive low-rank decomposition with less rounds, which could cause higher damage to the model accuracy which is unable to recover by fine-tuning. A suitable choice of ε is within $(0, 0.05)$ in our experiments.

5. Conclusion

In this paper, we proposed an iterative deep model compression and acceleration method in the frequency domain. We applied Discrete Cosine Transform (DCT) on convolutional kernels, and train them in the frequency domain to get their sparse weight matrices. We decomposed the frequency matrices using low-rank approximation, pruned and fine-tuned the model iteratively to restore the accuracy and obtain a compact model representation. We demonstrated that model inference can be conducted with the decomposed frequency matrices, where model parameters and inference cost can be reduced. We conducted extensive experiments using VGG and ResNet on three open datasets, which demonstrated that the proposed method outperforms the state-of-the-arts with significant practical improvements on model compression ratio and test accuracy.

Acknowledgments

This work was partially supported by the science and technology project from State Grid Corporation of China (No. 5700-202153172A-0-0-00), the National Natural Science Foundation of China (Grant Nos. 61972196, 61832008, 61832005), the Key R&D Program of Jiangsu Province, China (Grant No. BE2018116), the Collaborative Innovation Center of Novel Software Technology and Industrialization. The corresponding author is Sanglu Lu.

References

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks in the frequency domain. In *KDD*, pages 1475–1484, 2016.

Jia Deng, Wei Dong, Richard Socher, Li Jia Li, and Fei Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018a.

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

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 *ECCV*, pages 784–800, 2018b.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

A. Krizhevsky. Learning multiple layers of features from tiny images. In *Tech Report*, 2009.

Yann LeCun, John S. Denker, and Sara A. Solla. *Optimal brain damage*. Morgan Kaufmann Publishers Inc., 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NIPS*, pages 2181–2191, 2017.

Mingbao Lin, Rongrong Ji, Shaojie Li, Qixiang Ye, Yonghong Tian, Jianzhuang Liu, and Qi Tian. Filter sketch for network pruning. *arXiv preprint arXiv:2001.08514*, 2020a.

Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In *IJCAI*, 2020b.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, pages 2790–2799, 2019.

Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. In *NeurIPS*, pages 1043–1053, 2018.

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

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

Roger Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

Kr Rao, P Yip, K. R Rao, and K. R Rao. Discrete cosine transform: Algorithms, advantages, applications. *Discrete Cosine Transform Algorithms Advantages Applications*, 14 (6):507–508, 1990.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. In *IJCAI*, 2019.

Xavier Suau, Luca Zappella, and Nicholas Apostoloff. Network compression using correlation analysis of layer responses. *arXiv preprint arXiv:1807.10585*, 2018.

Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. Cop: Customized deep model compression via regularized correlation-based filter-level pruning. In *IJCAI*, 2019a.

Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. In *AAAI*, pages 12273–12280, 2019b.

Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Packing convolutional neural networks in the frequency domain. *TPAMI*, 41(10):2495–2510, 2018.

Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *CVPR*, 2020.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. In *ICLR*, 2020.

Tianyi Zhou and Dacheng Tao. Greedy bilateral sketch, completion & smoothing. *Journal of Machine Learning Research*, 2013.