Coordinating Filters for Faster Deep Neural Networks

Wei Wen
University of Pittsburgh
wew57@pitt.edu

Cong Xu
Hewlett Packard Labs
cong.xu@hpe.com

Chunpeng Wu
University of Pittsburgh
chw127@pitt.edu

Yandan Wang
University of Pittsburgh
yaw46@pitt.edu

Yiran Chen
Duke University
yiran.chen@duke.edu

Hai Li
Duke University
hai.li@duke.edu

Abstract

Very large-scale Deep Neural Networks (DNNs) have achieved remarkable successes in a large variety of computer vision tasks. However, the high computation intensity of DNNs makes it challenging to deploy these models on resource-limited systems. Some studies used low-rank approaches that approximate the filters by low-rank basis to accelerate the testing. Those works directly decomposed the pre-trained DNNs by Low-Rank Approximations (LRA). How to train DNNs toward lower-rank space for more efficient DNNs, however, remains as an open area. To solve the issue, in this work, we propose Force Regularization, which uses attractive forces to enforce filters so as to coordinate more weight information into lower-rank space[1]*. We mathematically and empirically verify that after applying our technique, standard LRA methods can reconstruct filters using much lower basis and thus result in faster DNNs. The effectiveness of our approach is comprehensively evaluated in ResNets, AlexNet, and GoogLeNet. In AlexNet, for example, Force Regularization gains* $2\times$ *speedup on modern GPU without accuracy loss and* $4.05\times$ *speedup on CPU by paying small accuracy degradation. Moreover, Force Regularization better initializes the low-rank DNNs such that the fine-tuning can converge faster toward higher accuracy. The obtained lower-rank DNNs can be further sparsified, proving that Force Regularization can be integrated with state-of-the-art sparsity-based acceleration methods.*

1. Introduction

Deep Neural Networks (DNNs) have achieved record-breaking accuracy in many image classification tasks [16][24][25][10]. With the advances of algorithms, availability of database, and improvement in hardware performance, the depth of DNNs grows dramatically from a few to hundreds or even thousands of layers, enabling human-level performance [9]. However, deploying these large models on resource-limited platforms, *e.g.,* mobiles and autonomous cars, is very challenging due to the high demand in the computation resource and hence energy consumption.

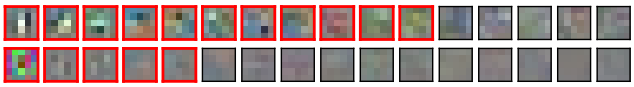

Figure 1. The low-rank basis of filters in the first layer of the convolutional neural network [16] on CIFAR-10. The low-rank basis is formed by the most significant principal filters that are obtained by PCA. Top: the low-rank basis of the original network. Bottom: the low-rank basis of the same network after applying *Force Regularization*. The number of red boxes indicates the required rank to reconstruct the original filters with $\leq 20\%$ error.

Recently, many techniques to accelerate the testing process of deployed DNNs have been studied, such as weight sparsifying or connection pruning [8][7][28][23][22][6][19]. These approaches require delicate hardware customization and/or software design to transfer sparsity into practical speedup. Unlike sparsity-based methods, *Low-Rank Approximation* (LRA) methods [22][4][5][12][11][26][27][18][30][14] directly decompose an original large model to a compact model with more lightweight layers. Thanks to the redundancy (correlation) among filters in DNNs, original weight tensors can be approximated by very low-rank basis. From the viewpoint of matrix computation, LRA approximates a large weight matrix by the product of two or more small ones to reduce computation complexity.

Previous LRA methods mostly focus on how to decompose the pre-trained weight tensors for maximizing the reduction of computation complexity, meanwhile retaining the classification accuracy. Instead, we propose to nudge the weights by additional gradients (*attractive forces*) to coordinate the filters to a more correlated state. Our approach

[1]The source code is available in https://github.com/wenwei202/caffe

aims to improve the correlation among filters and therefore obtain more lightweight DNNs through LRA. *To the best of our knowledge, this is the first work to train DNNs toward lower-rank space such that LRA can achieve faster DNNs.*

The motivation of this work is fundamental. It has been proven that trained filters are highly clustered and correlated [5][4][12]. Suppose each filter is reshaped as a vector. A cluster of highly-correlated vectors then will have small included angles. If we are able to coordinate these vectors toward a state with smaller included angles, the correlation of the filters within that cluster improves. Consequently, LRA can produce a DNN with lower ranks and higher computation efficiency.

We propose a *Force Regularization* to coordinate filters in DNNs. As demonstrated in Fig. 1, when using the same LRA method, say, cross-filter *Principal Component Analysis* (PCA) [30], applying *Force Regularization* can greatly reduce the required ranks from the original design (*i.e.*, 5 *vs.* 11), while keeping the same approximation errors ($\leq$ 20%). As we shall show in Section 5, applying *Force Regularization* in the training of state-of-the-art DNNs will successfully obtain lower-rank DNNs and thus improve computation efficiency, *e.g.*, 4.05$\times$ speedup for *AlexNet* with small accuracy loss.

The contributions of our work include: (1) We propose an effective and easy-to-implement *Force Regularization* to train DNNs for *lower*-rank approximation. To the best of our knowledge, this is the first work to manipulate the correlation among filters during training such that LRA can achieve faster DNNs; (2) DNNs manipulated by *Force Regularization* can have better initialization for the retraining of LRA-decomposed DNNs, resulting in faster convergence to better accuracy; (3) Those lightweight DNNs that have been aggressively compressed by our method can be further sparsified. That is, our method can be integrated with state-of-the-art sparsity-based methods to potentially achieve faster computation; (4) *Force Regularization* can be easily generalized to *Discrimination Regularization* that can learn more discriminative filters to improve classification accuracy; (5) Our implementation is open-source on both CPUs and GPUs.

2. Related work

Low-rank approximation. LRA method decomposes a large model to a compact one with more lightweight layers by weight/tensor factorization. Denil *et al.* [4] studied different dictionaries to remove the redundancy between filters and channels in DNNs. Jaderberg *et al.* [12] explored filter and data reconstruction optimizations to attain optimal separable basis. Denton *et al.* [5] clustered filters, extended LRA (*e.g.*, *Singular Value Decomposition*, SVD) to larger-scale DNNs, and achieved 2$\times$ speedup for the first two layers with 1% accuracy loss. Many new decomposition methods were proposed [11][26][18][30] and the effectiveness of LRA in state-of-the-art DNNs were evaluated [24][25]. Similar evaluations on mobile devices were also reported [14][27]. Unlike them, we propose *Force Regularization* to coordinate DNN filters to more correlated states, in which *lower*-rank or more compact DNNs are achievable for faster computation.

Sparse deep neural networks. The studies on sparse DNNs can be categorized into two types: non-structured [20][23][22][8][6] and structured [28][21][19][1] sparsity methods. The first category prunes each connection independently. Consequently, sparse weights are randomly distributed. The level of non-structured sparsity is usually insufficient to achieve good practical speedup in modern hardware [28][19]. Software optimization [23][22] and hardware customization [7] are proposed to overcome this issue. Conversely, the structured approaches prune connections group by group, such that the sparsified DNNs have regular distribution of sparse weights. The regularity is friendly to modern hardware for acceleration. Our work is orthogonal to sparsity-based methods. More importantly, we find that DNNs accelerated by our method can be further sparsified by both non-structured and structured sparsity methods, potentially achieving faster computation.

3. Correlated Filters and Their Approximation

The prior knowledge is that correlation exists among trained filters in DNNs and those filters lie in a low-rank space. For example, the color-agnostic filters [16] learned in the first layer of *AlexNet* lie in a hyper-plane, where RGB channels at each pixel have the same value. Fig. 2 presents the results of *Linear Discriminant Analysis* (LDA) of the first convolutional filters in *AlexNet* and *GoogLeNet*. The filters are normalized to unit vectors and colored to four clusters by k-means clustering, and then projected to 2D space by LDA to maximize cluster separation. The figure indicates high correlation among filters within a cluster. A naïve approach of filter approximation is to use the centroid of a cluster to approximate filters within that cluster, thus, the number of clusters is the rank of the space. Essentially, k-means clustering is a LRA [2] method, although we will

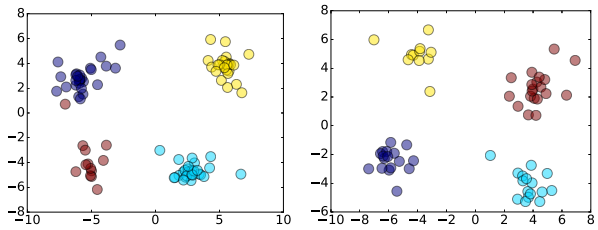

Figure 2. *Linear Discriminant Analysis* (LDA) of filters in the first convolutional layer of *AlexNet* (left) and *GoogLeNet* (right).

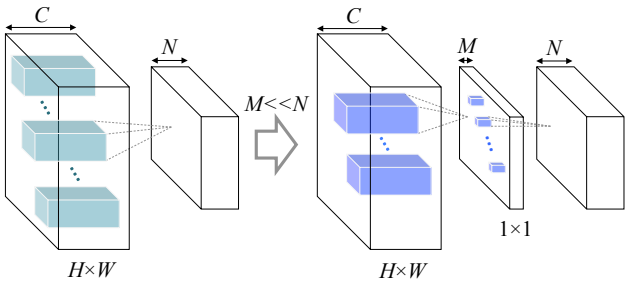

Figure 3. Cross-filter LRA of a convolutional layer.

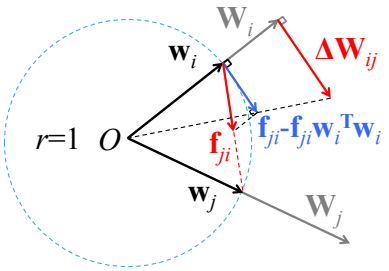

Figure 4. *Force Regularization* to coordinate filters.

later show that other LRA methods can give better approximation. The motivation of this work is that if we are able to nudge filters during the training such that the filters within a cluster are coordinated closer and some adjacent clusters are even merged into one cluster, then more accurate filter approximation using lower rank can be achieved. We propose *Force Regularization* to realize it.

Before introducing *Force Regularization*, we first mathematically formulate LRA of DNN filters. Theoretically, almost all LRA methods can gain lower-rank approximation upon our method because filters are coordinated to more correlated state. Instead of onerously replicating all of these LRA methods, we choose cross-filter approximation [4][30] and a state-of-the-art work in [26] as our baselines.

Fig. 3 illustrates the cross-filter approximation of a convolutional layer. We assume all weights in a convolutional layer is a tensor $\mathcal{W} \in \mathbb{R}^{N\times C\times H\times W}$, where N and C are the numbers of filters and input channels, and H and W are the spatial height and width of the filters, respectively. With input feature map $\mathcal{I}$, the n-th output feature map $\mathcal{O}_n = \mathcal{W}_n * \mathcal{I}$, where $\mathcal{W}_n \in \mathbb{R}^{1\times C\times H\times W}$ is the n-th filter. Because of the redundancy (or correlation) across the filters [4], tensor $\mathcal{W}_n (\forall n \in [1...N])$ can be approximated by a linear combination of the basis $\mathcal{B}_m \in \mathbb{R}^{1\times C\times H\times W} (m \in [1...M], M \ll N)$ of a low-rank space $\mathcal{B} \in \mathbb{R}^{M\times C\times H\times W}$, such as

$$\mathcal{O}_n \approx \left(\sum_{m=1}^{M} b_m^{(n)} \mathcal{B}_m\right) * \mathcal{I} = \sum_{m=1}^{M} \left(b_m^{(n)} \mathcal{F}_m\right). \quad (1)$$

Where $b_m^{(n)}$ is a scalar, and $\mathcal{F}_m = \mathcal{B}_m * \mathcal{I}$ is the feature map generated by basis filter $\mathcal{B}_m$. Therefore, the output feature map $\mathcal{O}_n$ is a linear combination of $\mathcal{F}_m (m \in [1...M])$ which can be interpreted as the feature map basis. Since the linear combination essentially is a 1×1 convolution, the convolutional layer can be decomposed to two sequential lightweight convolutional layers as shown in Fig. 3. The original computation complexity is $\mathcal{O}(NCHWH'W')$, where H' and W' is the height and width of output feature maps, respectively. After applying cross-filter LRA, the computation complexity is reduced to $\mathcal{O}(MCHWH'W' + NMH'W')$. The computation complexity decreases when the rank $M < \frac{NCHW}{CHW+N}$.

4. Force Regularization

4.1. Regularization by Attractive Forces

This section proposes *Force Regularization* from the perspective of physics. It is a gradient-based approach that adds extra gradients to data loss gradients. The data loss gradients aim to minimize classification error as traditional DNNs do. The extra gradients introduced by *Force Regularization* gently adjust the lengths and directions of data loss gradients so as to nudge filters to a more correlated state. With a good setup of hyper-parameter, our method can coordinate more useful information of filters to a lower-rank space meanwhile maintain accuracy. Inspired by Newton's Laws, we propose an intuitive, computation-efficient and effective *Force Regularization* that uses attractive forces to coordinate filters.

Force Regularization: As illustrated in Fig. 4, suppose the filter $\mathcal{W}_n \in \mathcal{W}$ is reshaped as a vector $\mathbf{W}_n \in \mathbb{R}^{1\times CHW}$ and normalized as $\mathbf{w}_n \in \mathbb{R}^{1\times CHW} (\forall n \in [1...N])$, with their origin at O. We introduce the pair-wise *attractive force* $\mathbf{f}_{ji} = f(\mathbf{w}_j - \mathbf{w}_i) (\forall i, j \in [1...N])$ on $\mathbf{w}_i$ generated by $\mathbf{w}_j$. The gradient of *Force Regularization* to update filter $\mathbf{W}_i$ is defined as

$$\Delta\mathbf{W}_i = \sum_{j=1}^{N} \Delta\mathbf{W}_{ij} = ||\mathbf{W}_i|| \sum_{j=1}^{N} \left(\mathbf{f}_{ji} - \mathbf{f}_{ji}\mathbf{w}_i^T\mathbf{w}_i\right), \quad (2)$$

where $||\cdot||$ is the Euclidean norm. The regularization gradient in Eq. (2) is perpendicular to filter vector and can be efficiently computed by addition and multiplication. The final updating of weights by gradient descent is

$$\mathbf{W}_i \leftarrow \mathbf{W}_i - \eta \cdot \left(\frac{\partial E(\mathcal{W})}{\partial \mathbf{W}_i} - \lambda_s \cdot \Delta\mathbf{W}_i\right), \quad (3)$$

where $E(\mathcal{W})$ is data loss, η is learning rate and $\lambda_s > 0$ is the coefficient of *Force Regularization* to trade off the rank and accuracy. We select λ_s by cross-validation in this work. The gradient of common weight-wise regularization (*e.g.,* ℓ_2-norm) is omitted in Eq. (3) for simplicity.

Fig. 4 intuitively explains our method. Suppose each vector $\mathbf{w}_i$ is a rigid stick and there is a particle fixed at the endpoint. The particle has unit mass, and the stick is massless and can freely spin around the origin. Given the pair-wise attractive forces (*e.g.,* universal gravitation) $\mathbf{f}_{ji}$, Eq. (2) is the acceleration of particle i. As the forces are attractive, neighbor particles tend to spin around the origin to assemble together. Although our regularizer seems to collapse all particles to one point which is the rank-one space for most lightweight DNNs, there exist gradients of data loss to avoid this. More specific, pre-trained filters orient to discriminative directions $\mathbf{w}_n$ $(n \in [1...N])$. In each direction $\mathbf{w}_n$, there are some correlated filters as observed in Fig. 2. During the subsequent retraining with our regularizer, regularization gradients coordinate a cluster of filters closer to a typical direction $\mathbf{d}_m$ $(m \in [1...M], M \ll N)$, but data loss gradients avoid collapsing $\mathbf{d}_m$ together so as to maintain the filters' capability of extracting discriminative features. If all filters could be extremely collapsed toward one point meanwhile maintain classification accuracy, it implies the filters are over-redundant and we can attain a very efficient DNN by decomposing it to a rank-one space.

We derive the *Force Regularization* gradient from the *normalized* filters based on the following facts: (1) A normalized filter is on the unit hypersphere, and its orientation is the only free parameter we need to optimize; (2) The gradient of $\mathbf{W}_i$ can be easily scaled by the vector length $||\mathbf{W}_i||$ without changing the angular velocity.

In Eq. (2), $\mathbf{f}_{ji} = f(\mathbf{w}_j - \mathbf{w}_i)$ is the force function related to distance. We study ℓ_2*-norm Force*

$$f_{\ell_2}(\mathbf{w}_j - \mathbf{w}_i) = \mathbf{w}_j - \mathbf{w}_i \tag{4}$$

and ℓ_1*-norm Force*

$$f_{\ell_1}(\mathbf{w}_j - \mathbf{w}_i) = \frac{\mathbf{w}_j - \mathbf{w}_i}{||\mathbf{w}_j - \mathbf{w}_i||} \tag{5}$$

in this work. We define the force of Eq. (4) as ℓ_2*-norm Force* because the strength linearly decreases with the distance $||\mathbf{w}_j - \mathbf{w}_i||$, just as the gradient of regularization ℓ_2-norm does. We name the force of Eq. (5) as ℓ_1*-norm Force* because the gradient is a constant unit vector regardless of the distance, just as the gradient of sparsity regularization ℓ_1-norm is.

4.2. Mathematical Implications

This section explains the mathematical implications behind: *Force Regularization* is related to but *different* from minimizing the sum of pair-wise distances between normalized filters.

Theorem 1 *Suppose filter $\mathcal{W}_n \in \mathcal{W}$ is reshaped as a vector $\mathbf{W}_n \in \mathbb{R}^{1\times CHW}$ and normalized as $\mathbf{w}_n \in \mathbb{R}^{1\times CHW}$ $(\forall n \in [1...N])$. For each filter, Force Regularization under ℓ_2-norm force has the same gradient direction of regularization $R(\mathcal{W})$, but differs by adapting the step size to the filer's length, where*

$$R(\mathcal{W}) = \frac{1}{2}\sum_{j=1}^{N}\sum_{i=1}^{N}\left|\left|\frac{\mathbf{W}_j}{||\mathbf{W}_j||} - \frac{\mathbf{W}_i}{||\mathbf{W}_i||}\right|\right|^2. \tag{6}$$

Proof: Because $\mathbf{w}_j = \frac{\mathbf{W}_j}{||\mathbf{W}_j||}$,

$$\begin{aligned}\frac{\partial R(\mathcal{W})}{\partial \mathbf{W}_i} &= \frac{1}{2}\sum_{j=1}^{N}\frac{\partial\left(\mathbf{w}_j - \mathbf{w}_i\right)\left(\mathbf{w}_j - \mathbf{w}_i\right)^T}{\partial \mathbf{W}_i} \\ &= \frac{1}{2}\sum_{j=1}^{N}\frac{\partial\left(1 - 2\mathbf{w}_j\mathbf{w}_i^T + 1\right)}{\partial \mathbf{W}_i} \\ &= -\sum_{j=1}^{N}\frac{\partial\left(\mathbf{w}_j\mathbf{w}_i^T\right)}{\partial \mathbf{W}_i} = -\sum_{j=1}^{N}\mathbf{w}_j\frac{\partial \mathbf{w}_i^T}{\partial \mathbf{W}_i},\end{aligned} \tag{7}$$

where $\frac{\partial \mathbf{w}_i^T}{\partial \mathbf{W}_i} := \mathbf{G}_i$ is a derivative matrix with element

$$\begin{aligned}G_i^{(pq)} &= \frac{\partial w_i^{(p)}}{\partial W_i^{(q)}} = \frac{\partial \frac{W_i^{(p)}}{||\mathbf{W}_i||}}{\partial W_i^{(q)}} \\ &= \frac{1}{||\mathbf{W}_i||}\left(\delta(p,q) - \frac{W_i^{(p)}W_i^{(q)}}{||\mathbf{W}_i||^2}\right).\end{aligned} \tag{8}$$

Superscripts $p, q \in [1 \dots CHW]$ index the elements in vectors $\mathbf{w}_i$ and $\mathbf{W}_i$. $\delta(p,q)$ is the *unit impulse* function:

$$\delta(p,q) = \begin{cases}1 & p = q \\ 0 & p \neq q\end{cases}. \tag{9}$$

Therefore,

$$\mathbf{G}_i = \frac{1}{||\mathbf{W}_i||}\left(\mathbf{I} - \mathbf{w}_i^T\mathbf{w}_i\right). \tag{10}$$

Replacing Eq. (10) to Eq. (7), we have

$$\begin{aligned}-\frac{\partial R(\mathcal{W})}{\partial \mathbf{W}_i} &= \frac{1}{||\mathbf{W}_i||}\sum_{j=1}^{N}\left(\left(\mathbf{w}_j - \mathbf{w}_i\right) - \left(\mathbf{w}_j - \mathbf{w}_i\right)\mathbf{w}_i^T\mathbf{w}_i\right) \\ &= \frac{1}{||\mathbf{W}_i||}\left(\left(\sum_{j=1}^{N}\mathbf{f}_{ji}\right) - \left(\sum_{j=1}^{N}\mathbf{f}_{ji}\right)\mathbf{w}_i^T\mathbf{w}_i\right),\end{aligned} \tag{11}$$

Table 1. Ranks *vs.* scalers of step sizes of regularization gradients.

Scaler	Error	conv1*	conv2	conv3
0 (baseline)	18.0%	17/32	27/32	55/64
$\|\|\mathbf{W}_i\|\|$	17.9%	15/32	22/32	30/64
$1/\|\|\mathbf{W}_i\|\|$	18.0%	16/32	27/32	32/64

* The first convolutional layer.

where $\mathbf{f}_{ji} = f_{\ell_2}(\mathbf{w}_j - \mathbf{w}_i) = \mathbf{w}_j - \mathbf{w}_i$. Therefore, Eq. (11) and Eq. (2) have the same direction.

Theorem 1 states that our proposed *Force Regularization* in Eq. (2) is related to Eq. (11). However, the step size of the gradient in Eq. (2) is scaled by the length $||\mathbf{W}_i||$ of the filter instead of its reciprocal in Eq. (11). This ensures that the filter spins the same angle regardless of its length and avoids the issue of being divided by zero. Table 1 summarizes the ranks *vs.* step sizes for the *ConvNet* [16], which is trained by CIFAR-10 database without data augmentation. The original *ConvNet* has 32, 32, and 64 filters in each convolutional layer, respectively. The rank is the smallest number of basis filters (in Fig. 3) obtained by PCA with $\leq 5\%$ reconstruction error. Therefore, $||\mathbf{W}_i||$ works better than its reciprocal when coordinating filters to a lower-rank space.

Following the same proof procedure, we can easily find that *Force Regularization* under ℓ_1*-norm Force* has the same conclusion when

$$R(\mathcal{W}) = \sum_{j=1}^{N} \sum_{i=1}^{N} \left|\left| \frac{\mathbf{W}_j}{||\mathbf{W}_j||} - \frac{\mathbf{W}_i}{||\mathbf{W}_i||} \right|\right|. \qquad (12)$$

5. Experiments

5.1. Implementation

Our experiments are performed in Caffe [13] using CIFAR-10 [15] and ILSVRC-2012 ImageNet [3]. Published models are adopted as the baselines: In CIFAR-10, we choose *ConvNet* without data augmentation [16] and *ResNets-20* with data augmentation [10]. We adopt the same shortcut connections in [28] for *ResNets-20*. For ImageNet, we use *AlexNet* and *GoogLeNet* models trained by Caffe, and report accuracy using only center crop of images.

Our experiments of *Force Regularization* show that, with the same maximum iterations, the training from the baseline can achieve a better tradeoff between accuracy and speedup comparing with the training from scratch, because the baseline offers a good initial point for both accuracy and filter correlation. During the training with *Force Regularization* on CIFAR-10, we use the same base learning rate as the baseline; while in ImageNet, $0.1\times$ base learning rate of the baseline is adopted.

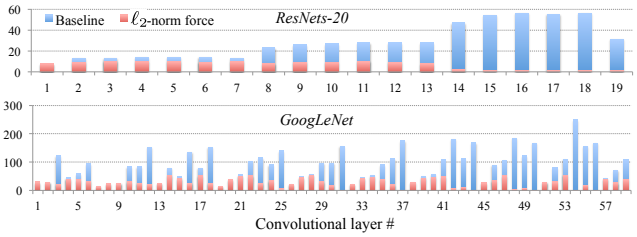

Figure 5. The *rank* M in each convolutional layer of *ResNets-20* and *GoogLeNet*. Red bar overlaps blue bar. The accuracy loss is 0.75% for *ResNets-20* and 2.46% (top-5) for *GoogLeNet*.

5.2. Rank Analysis of Coordinated DNNs

In light of various low-rank approximation methods, without losing the generalization, we first adopt *Principal Component Analysis* (PCA) [30][22] to evaluate the effectiveness of *Force Regularization*. Specifically, the filter tensor $\mathcal{W}$ can be reshaped to a matrix $\mathbf{W} \in \mathbb{R}^{N \times CHW}$, the rows of which are the reshaped filters $\mathbf{W}_n$ $(\forall n \in [1...N])$. PCA minimizes the *least square reconstruction error* when projecting a column $(\mathbb{R}^N)$ of $\mathbf{W}$ to a low-rank space $\mathbb{R}^M (M \ll N)$. The reconstruction error is $e_M = \sum_{i=M+1}^{N} \lambda_i$, where λ_i is the i-th largest eigenvalue of covariance matrix $\frac{\mathbf{W}\mathbf{W}^T}{CHW-1}$. Under the constraint of *error percentage* $\frac{e_M}{e_0}$ (*e.g.*, $\frac{e_M}{e_0} \leq 5\%$), *lower*-rank approximation can be obtained if the minimal rank M can be *smaller*. In this section, without explicit explanation, we define *rank* M of a convolutional layer as the minimal M which has $\leq 5\%$ reconstruction error by PCA.

Table 2 summarizes the *rank* M in each layer of *ConvNet* and *AlexNet* without accuracy loss after *Force Regularization*. In the baselines, the learned filters in the front layers are intrinsically in a very low-rank space but the *rank* M in deeper layers is high. This could explain why only speedups of the first two convolutional layers were reported in [5]. Fortunately, by using either ℓ_2-norm or ℓ_1-norm force, our method can efficiently maintain the low *rank* M in the first two layers (*e.g.*, conv1-conv2 in *AlexNet*), meanwhile significantly reduce the *rank* M of deeper layers (*e.g.*, conv3-conv5 in *AlexNet*). On average, our method can reduce the layer-wise rank ratio by $\sim 50\%$. The effectiveness of our method on deep layers is very important as the

Table 2. The *rank M* in each convolutional layer after *Force Regularization*.

Net	Force	Top-1 error	conv1	conv2	conv3	conv4	conv5	Average rank ratio ‡
ConvNet	None (baseline)†	18.0%	17/32‡	27/32	55/64	–	–	74.48%
ConvNet	ℓ_2-norm	17.9%	15/32	22/32	30/64	–	–	54.17%
ConvNet	ℓ_1-norm	18.0%	17/32	25/32	20/64	–	–	54.17%
AlexNet	None (baseline)	42.63%	47/96	164/256	306/384	318/384	220/256	72.29%
AlexNet	ℓ_2-norm	42.70%	49/96	143/256	128/384	122/384	161/256	46.98%
AlexNet	ℓ_1-norm	42.45%	49/96	155/256	157/384	108/384	178/256	50.03%

†The baseline without *Force Regularization*. ‡M/N: Low *rank* M over full rank N, which is defined as rank ratio.

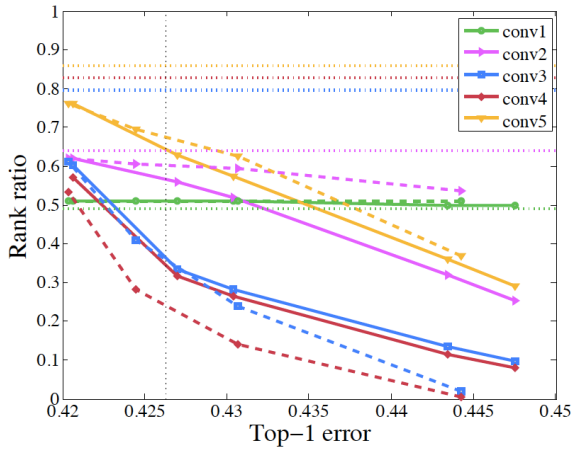

Figure 6. Th rank ratio (having $\leq 5\%$ PCA reconstruction error) in each layer *vs.* top-1 error for *AlexNet*. Horizontal dotted lines represent the rank ratios of the baseline, and vertical dotted line is the error of baseline. Solid (dashed) curves depict rank ratios of the *AlexNet* after *Force Regularization* by ℓ_2-norm (ℓ_1-norm) force. Each layer is denoted by a typical color. The sensitivity of hyper-parameter λ_s: along the direction from left to right, λ_s of ℓ_2-norm force changes from 1.2e-5, to 1.8e-5, 2.0e-5, 3.0e-5, and 3.5e-5; and for ℓ_1-norm force, it changes from 1.5e-5, to 1.8e-5, 2.0e-5, and 2.5e-5.

depth of modern DNNs grows dramatically [25][10]. Fig. 5 shows the *rank* M of *ResNets-20* [10] and *GoogLeNet* [25] after *Force Regularization*, representing the scalability of our method on deeper DNNs. With an acceptable accuracy loss, 5 layers in *ResNets-20* and 6 layers in *GoogLeNet* are even coordinated to *rank* $M = 1$, which indicates those Inception blocks in *GoogLeNet* or Residual blocks in *ResNets* have been over-parameterized and can be greatly simplified.

To study the trade-off between rank, accuracy, and the pros and cons of ℓ_2-norm and ℓ_1-norm force, we conducted comprehensive experiments on *AlexNet*. As shown in Fig. 6, with mere 1.71% (1.80%) accuracy loss, the average rank ratio can be reduced to 28.59% (28.72%) using ℓ_2-norm (ℓ_1-norm) force. Very impressively, the *rank* M of each group in conv4 can be reduced to one by ℓ_1-norm force. The results also show that ℓ_2-norm force is more effective than ℓ_1-norm force when the rank ratio is high (*e.g.*, conv2 and conv5), while ℓ_1-norm force works better for layers whose potential rank ratios are low (*e.g.*, conv3 and conv4). In general, ℓ_2-norm force can better balance the ranks across all the layers.

Because *Force Regularization* coordinates more useful weight information in a low-rank space, it essentially can provide a better training initialization for the DNNs that are decomposed by LRA. Fig. 7 plots the training data loss and top-1 validation error of *AlexNet*, which is decomposed to the same ranks by PCA. The baseline is the original *AlexNet* and the other *AlexNet* is coordinated by *Force Regularization*. The figure shows that the error sharply converges to a low level after a few iterations, indicating LRA provides a very good initialization for the low-rank DNNs. Training it from scratch has significant accuracy loss. More importantly, DNNs coordinated by *Force Regularization* can converge faster to a lower error.

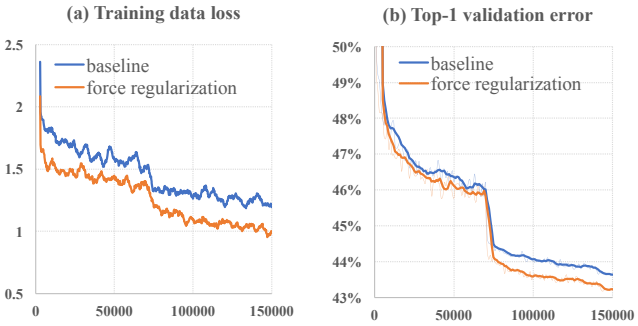

Figure 7. Training data loss and top-1 validation error vs. iteration when fine-tuning *AlexNet* which is decomposed to the same ranks.

Besides PCA [22][30], we also evaluated the effectiveness of *Force Regularization* when integrating it with SVD [5][26] or k-means clustering [5][2]. Table 3 compares the accuracies of *AlexNet* decomposed by different LRA methods. All LRAs preserve the same ranks in all layers, which means the decomposed *AlexNet* have the same network structure. In summary, PCA and SVD obtain similar accuracy and surpass k-means clustering. Due to the limited pages, we adopt PCA as the representative in our study.

5.3. Acceleration of DNN Testing

In our experiments, we first train DNNs with *Force Regularization*, then decompose DNNs using LRA methods and fine-tune them to recover accuracy. In evaluation of speed, we omit small CIFAR-10 database and focus on large-scale DNNs on ImageNet, whose speed is a real concern. To prove the effective acceleration of *Force Regularization*, we adopt the speedup of state-of-the-art LRAs [30][4][26] as our baseline. Our speedup is achieved in the case that the DNN filters are first coordinated by *Force Regularization* and then decomposed using the same LRAs. The practical GPU speed is profiled by the advanced hardware (NVIDIA

Table 3. The accuracy of different LRA under the same ranks.

Force	LRA	Top-1 error
None	PCA	43.21%
	SVD†	43.27%
	k-means†	44.34%
ℓ_2-norm	PCA	43.25%
	SVD†	43.20%
	k-means†	44.80%

† SVD and k-means preserve the same ranks with PCA

Table 4. The higher speedups of *AlexNet* by *Force Regularization*.

Force	Top-1 error		conv3	conv4	conv5
None	43.21%	rank	184	201	146
ℓ_2-norm	43.25%	rank	124	106	129
None	43.21%	GPU	1.58×	1.21×	1.15×
ℓ_2-norm	43.25%	GPU	2.16×	2.03×	1.33×
None	43.21%	CPU	1.78×	1.60×	1.47×
ℓ_2-norm	43.25%	CPU	2.45×	2.76×	1.64×
None	43.21%	theoretical	1.79×	1.72×	1.63×
ℓ_2-norm	43.25%	theoretical	2.65×	3.26×	1.85×

GTX 1080) and software (cuDNN 5.0). The CPU speed is measured in Intel Xeon E5-2630 and ATLAS library. The batch size is 256.

Cross-filter LRA: We first evaluate the speedup of cross-filter LRA shown in Fig. 3. In previous works [5][26], the optimal rank in each layer can be selected layer-by-layer using cross validation. However, the number of hyper-parameters increases linearly with the depth of DNNs. To save development time, we utilize an identical *error percentage* $\frac{e_M}{e_0}$ across all layers as the single hyper-parameter although layer-wise rank selection may give better tradeoff. The rank in a layer is the minimal M which has error $\leq \frac{e_M}{e_0}$.

As aforementioned in Section 5.2 and Table 2, the learned conv1 and conv2 of *AlexNet* are already in a very low-rank space and achieve good speedups using LRAs [5]. Thus we mainly focus on conv3-conv5 here. Table 4 summarizes the speedups of PCA approximation of *AlexNet* with and without ℓ_2-norm *Force Regularization*. With ignoble accuracy difference, *Force Regularization* successfully coordinates filters to a lower-rank space and accelerates the testing by a higher factor, comparing with the state-of-the-art LRA. Similar results are observed when applying ℓ_1-norm force.

Results in Table 4 also show that practical speedup is different from theoretical speedup. Generally, the difference is smaller in lower-performance processors. In CPU mode of Table 4, *Force Regularization* achieves 2× speedup of total convolutional time.

Speeding up state-of-the-art LRA: We also duplicate the state-of-the-art work [26] as the baseline[2] (lra_1). After LRA, *AlexNet* is fine-tuned with learning rate starting from 0.001 and divided by 10 at iteration 70,000 and 140,000. Fine-tuning terminates after 150,000 iterations.

The first row in Table 5 contains the results of the baseline [26], which don't scale well to the advanced "TITAN 1080 + cuDNN 5.0" in conv3–5. This is because 3×3 convolution is highly optimized in cuDNN 5.0, *e.g.*, using Winograd's minimal filtering algorithms [17]. However, the baseline decomposes the 3×3 convolution to a pair of

[2]Code is provided by the authors in https://github.com/chengtaipu/lowrankcnn/

Table 5. The higher speedup factors by force regularization.

LRA	Force	Top-5 err.		conv3	conv4	conv5
lra_1 [26]	None	20.65%	GPU	0.86×	0.57×	0.40×
lra_2	None	19.93%	GPU	1.89×	1.57×	1.57×
lra_2	ℓ_2-norm	20.14%	GPU	2.25×	2.03×	1.60×
lra_2	ℓ_2-norm	21.68%	GPU	3.56×	3.01×	2.40×
			CPU	4.81×	4.00×	2.92×

3×1 and 1×3 convolution so that the optimized cuDNN is not fully exploited. This will be a common issue in the baseline, considering Winograd's algorithm is universally used and 3×3 convolution is one of the most common structures. We find that LRA in Fig. 3 can be utilized for conv 3–5 to solve this issue, because it can maintain the 3×3 shape. We name this LRA as lra_2, which decomposes conv1–conv2 using LRA in [26] and conv 3–5 using LRA of Fig. 3. The second row in Table 5 shows that our lra_2 can scale well to the hardware and software advances of "TITAN 1080 + cuDNN 5.0". More importantly, *Force Regularization* on conv3–5 can enforce them to more lightweight layers and attain higher speedup factors than lra_2 without using it. The result is shown in the third row, which in total achieves 2.03× speedup for the whole convolution in GPU. With small accuracy loss in row 4 of Table 5, *Force Regularization* achieves 2.50× speedup of total convolution on GPU and 4.05× on CPU.

Table 6 compares our method with state-of-the-art DNN acceleration methods, in CPU mode. When the speedup of total time was not reported by the authors, we estimate it by the weighted average speedups over all layers, where the weighting coefficients are derived from the percentage of running time of each layer. In our hardware platform, conv1–conv5 respectively consume 15.89%, 28.25%, 24.32%, 18.70% and 12.84% testing time. The estimation is accurate, for example, we estimate 2.58× of total time in *one-shot* [14], which is very close to 2.52× reported by the authors. Comparing with both *cp-decomposition* and *one-shot* methods, our method can achieve higher accuracy and higher speedup. Comparing with *SSL*, with almost the same top-5 error (21.68% vs. 21.63%), we can attain higher speedup of 4.05× *vs.* 3.13×.

deep-compression [7] reported 3× to 4× speedups in fully-connected layers when batch size was 1. However, convolution is the bottleneck of DNNs, *e.g.*, the convolution time in *AlexNet* is 5× of the time in fully-connected layers when profiled in our CPU platform. Moreover, no speedup was observed in the batching scenario as reported by the authors [7]. More importantly, as we will show in Section 5.4, our work can work together with sparsity-based methods (*e.g.*, *SSL* or *deep-compression*) to obtain *lower-rank and sparse DNNs* and potentially further accelerate the testing of DNNs.

Table 6. Comparison of speedup factor on *AlexNet* by state-of-the-art DNN acceleration methods.

Method	Top-5 err.	conv1	conv2	conv3	conv4	conv5	total
AlexNet in Caffe	19.97%	1.00×	1.00×	1.00×	1.00×	1.00×	**1.00**×
cp-decomposition [18]	20.97% (+1.00%)	–	4.00×	–	–	–	**1.27**×
one-shot [14]	21.67% (+1.70%)	1.48×	2.30×	3.84×	3.53×	3.13×	**2.52**×
SSL [28]	19.58% (-0.39%)	1.00×	1.27×	1.64×	1.68×	1.32×	**1.35**×
	21.63% (+1.66%)	1.05×	3.37×	6.27×	9.73×	4.93×	**3.13**×
our lra_2	20.14% (+0.17%)	2.61×	6.06×	2.48×	2.20×	1.58×	**2.69**×
	21.68% (+1.71%)	2.65×	6.22×	4.81×	4.00×	2.92×	**4.05**×

5.4. Lower-rank and Sparse DNNs

We sparsify the lightweight deep neural network (*i.e.*, the first one of lra_2 in Table 6), using Structured Sparsity Learning *SSL* [28] or non-structured *connection-pruning* [23]. Note that Guided Sparsity Learning (*GSL*) is not adopted in our *connection-pruning* though better sparsity is achievable when applying it. Figure 8 summarizes the results.

Experiments prove that our method can work together with both structured and non-structured sparsity methods to further compress and accelerate models. Comparing with *deep-compression* in Figure 8(a), our model has comparable compression rates but 2.69× faster testing time. Typically, our model has higher compression rates in convolutional layers, which provides more space for computation reduction and generalizes better to modern DNNs (*ResNets-152* [10], for example, whose parameters in fc layers are only 4%). In Figure 8(b), our accelerated model can be further accelerated using *SSL*. The shape-wise sparsity in conv3–5 of our model is slightly lower because our model is already aggressively compressed by LRA. The higher filter-wise sparsity, however, implies the orthogonality of our approach to *SSL*.

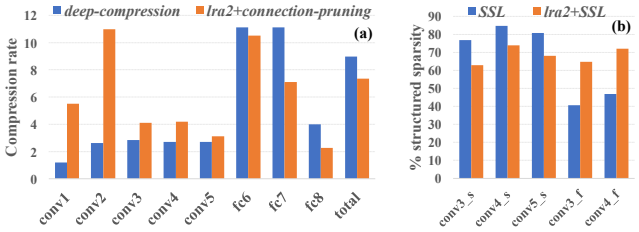

Figure 8. The results of sparsifying lightweight DNNs whose filters are coordinated to a lower-rank space by *Force Regularization*. In terms of *deep-compression* in (a), we only count the compression rate obtained from connection pruning for a fair comparison, but quantization and Huffman coding can also be utilized to improve the compression rate for our model. Based on *SSL* in (b), we enforce shape-wise sparsity on conv3_s, conv4_s and conv5_s to learn the shapes of basis filters meanwhile enforce filter-wise sparsity on conv3_f and conv4_f to learn the number of filters [28]. As each convolutional layer in the lra_2 is decomposed to two small layers, we respectively denote the first and second small layer by suffixing "_s" and "_f". The baseline and our model have the same accuracy.

5.5. Generalization of Force Regularization

In convolutional layers, each filter basically extracts a discriminative feature, *e.g.*, an orientation-selective pattern or a color blob in the first layer [16] or a high-level feature (*e.g.*, textures, faces, *etc.*) in deeper layers [29]. The discrimination among filters is important for classification performance. Our method can coordinate filters for more lightweight DNNs meanwhile maintain the discrimination. It can also be generalized to learn more discriminative filters to improve the accuracy. The extension to *Discrimination Regularization* is straightforward but effective: the opposite gradient of *Force Regularization* (*i.e.*, $\lambda_s < 0$) is utilized to update the filter. In this scenario, it works as the *repulsive force* to repel surrounding filters and enhance the discrimination. Table 7 summarizes the improved accuracy of state-of-the-art DNNs.

Acknowledgments

This work was supported in part by NSF CCF-1744082. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or their contractors.

Table 7. Improved accuracy with *Discrimination Regularization*.

Net	Regularization	Top-1 error
AlexNet	None (baseline)	42.63%
AlexNet	ℓ_2-norm force	41.71%
AlexNet	ℓ_1-norm force	41.53%
ResNets-20	None (baseline)	8.82%
ResNets-20	ℓ_2-norm force	7.97%
ResNets-20	ℓ_1-norm force	8.02%

References

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2262–2270, 2016. 2

[2] C. Bauckhage. k-means clustering is matrix factorization. *arXiv:1512.07548*, 2015. 2, 6

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 5

[4] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*. 2013. 1, 2, 3, 6

[5] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*. 2014. 1, 2, 5, 6, 7

[6] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems (NIPS)*. 2016. 1, 2

[7] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015. 1, 2, 7

[8] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*. 2015. 1, 2

[9] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, 2015. 1

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 5, 6, 8

[11] Y. Ioannou, D. P. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv:1511.06744*, 2015. 1, 2

[12] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014. 1, 2

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014. 5

[14] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv:1511.06530*, 2015. 1, 2, 7, 8

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 5

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2012. 1, 2, 5, 8

[17] A. Lavin. Fast algorithms for convolutional neural networks. *arXiv:1509.09308*, 2015. 7

[18] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv:1412.6553*, 2014. 1, 2, 8

[19] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2

[20] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 598–605, 1989. 2

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017. 2

[22] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 2, 5, 6

[23] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 2, 8

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 1, 2

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 2, 6

[26] C. Tai, T. Xiao, X. Wang, and W. E. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations (ICLR)*, 2016. 1, 2, 3, 6, 7

[27] P. Wang and J. Cheng. Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 2016 ACM on Multimedia Conference*, 2016. 1, 2

[28] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2016. 1, 2, 5, 8

[29] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014. 8

[30] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, Oct 2016. 1, 2, 3, 5, 6