Efficient and Accurate Approximations of Nonlinear Convolutional Networks

Xiangyu Zhang[1*] Jianhua Zou[1] Xiang Ming[1*] Kaiming He[2] Jian Sun[2]
[1]Xi'an Jiaotong University [2]Microsoft Research

Abstract

This paper aims to accelerate the test-time computation of deep convolutional neural networks (CNNs). Unlike existing methods that are designed for approximating linear filters or linear responses, our method takes the nonlinear units into account. We minimize the reconstruction error of the nonlinear responses, subject to a low-rank constraint which helps to reduce the complexity of filters. We develop an effective solution to this constrained nonlinear optimization problem. An algorithm is also presented for reducing the accumulated error when multiple layers are approximated. A whole-model speedup ratio of 4× is demonstrated on a large network trained for ImageNet, while the top-5 error rate is only increased by 0.9%. Our accelerated model has a comparably fast speed as the "AlexNet" [11], but is 4.7% more accurate.

1. Introduction

This paper addresses efficient test-time computation of deep convolutional neural networks (CNNs) [12, 11]. Since the success of CNNs [11] for large-scale image classification, the accuracy of the newly developed CNNs [24, 17, 8, 18, 19] has been continuously improving. However, the computational cost of these networks (especially the more accurate but larger models) also increases significantly. The expensive test-time evaluation of the models can make them impractical in real-world systems. For example, a cloud service needs to process thousands of new requests per seconds; portable devices such as phones and tablets mostly have CPUs or low-end GPUs only; some recognition tasks like object detection [4, 8, 7] are still time-consuming for processing a single image even on a high-end GPU. For these reasons and others, it is of practical importance to accelerate the test-time computation of CNNs.

There have been a few studies on approximating deep CNNs for accelerating test-time evaluation [22, 3, 10]. A commonly used assumption is that the convolutional filters are approximately low-rank along certain dimensions. So the original filters can be approximately decomposed into a series of smaller filters, and the complexity is reduced. These methods have shown promising speedup ratios on a single [3] or a few layers [10] with some degradation of accuracy.

The algorithms and approximations in the previous work are developed for reconstructing linear filters [3, 10] and linear responses [10]. However, the nonlinearity like the Rectified Linear Units (ReLU) [14, 11] is not involved in their optimization. Ignoring the nonlinearity will impact the quality of the approximated layers. Let us consider a case that the filters are approximated by reconstructing the linear responses. Because the ReLU will follow, the model accuracy is more sensitive to the reconstruction error of the positive responses than to that of the negative responses.

Moreover, it is a challenging task of accelerating the whole network (instead of just one or a very few layers). The errors will be accumulated if several layers are approximated, especially when the model is deep. Actually, in the recent work [3, 10] the approximations are applied on a single layer of large CNN models, such as those trained on ImageNet [2, 16]. It is insufficient for practical usage to speedup one or a few layers, especially for the deeper models which have been shown very accurate [18, 19, 8].

In this paper, a method for accelerating *nonlinear* convolutional networks is proposed. It is based on minimizing the reconstruction error of *nonlinear* responses, subject to a low-rank constraint that can be used to reduce computation. To solve the challenging constrained optimization problem, we decompose it into two feasible subproblems and iteratively solve them. We further propose to minimize an asymmetric reconstruction error, which effectively reduces the accumulated error of multiple approximated layers.

We evaluate our method on a 7-convolutional-layer model trained on ImageNet. We investigate the cases of accelerating each single layer and the whole model. Experiments show that our method is more accurate than the recent method of Jaderberg *et al.*'s [10] under the same speedup ratios. A *whole-model* speedup ratio of 4× is demonstrated, and its degradation is merely 0.9%. When our model is accelerated to have a comparably fast speed as the "AlexNet" [11], our accuracy is 4.7% higher.

*This work is done when Xiangyu Zhang and Xiang Ming are interns at Microsoft Research.

2. Approaches

2.1. Low-rank Approximation of Responses

Our observation is that the response at a position of a convolutional feature map approximately lies on a low-rank subspace. The low-rank decomposition can reduce the complexity. To find the approximate low-rank subspace, we minimize the reconstruction error of the responses.

More formally, we consider a convolutional layer with a filter size of $k \times k \times c$, where k is the spatial size of the filter and c is the number of input channels of this layer. To compute a response, this filter is applied on a $k \times k \times c$ volume of the layer input. We use $\mathbf{x} \in \mathbb{R}^{k^2c+1}$ to denote a vector that reshapes this volume (appending one as the last entry for the bias). A response $\mathbf{y} \in \mathbb{R}^d$ at a position of a feature map is computed as:

$$\mathbf{y} = \mathrm{W}\mathbf{x}. \tag{1}$$

where W is a d-by-(k^2c+1) matrix, and d is the number of filters. Each row of W denotes the reshaped form of a $k \times k \times c$ filter (appending the bias as the last entry). We will address the nonlinear case later.

If the vector $\mathbf{y}$ is on a low-rank subspace, we can write $\mathbf{y} = \mathrm{M}(\mathbf{y} - \bar{\mathbf{y}}) + \bar{\mathbf{y}}$, where M is a d-by-d matrix of a rank $d' < d$ and $\bar{\mathbf{y}}$ is the mean vector of responses. Expanding this equation, we can compute a response by:

$$\mathbf{y} = \mathrm{MW}\mathbf{x} + \mathbf{b}, \tag{2}$$

where $\mathbf{b} = \bar{\mathbf{y}} - \mathrm{M}\bar{\mathbf{y}}$ is a new bias. The rank-d' matrix M can be decomposed into two d-by-d' matrices P and Q such that $\mathrm{M} = \mathrm{PQ}^\top$. We denote $\mathrm{W}' = \mathrm{Q}^\top\mathrm{W}$ as a d'-by-(k^2c+1) matrix, which is essentially a new set of d' filters. Then we can compute (2) by:

$$\mathbf{y} = \mathrm{PW}'\mathbf{x} + \mathbf{b}. \tag{3}$$

The complexity of using Eqn.(3) is $O(d'k^2c) + O(dd')$, while the complexity of using Eqn.(1) is $O(dk^2c)$. For many typical models/layers, we usually have $O(dd') \ll O(d'k^2c)$, so the computation in Eqn.(3) will reduce the complexity to about d'/d.

Fig. 1 illustrates how to use Eqn.(3) in a network. We replace the original layer (given by W) by two layers (given by W′ and P). The matrix W′ is actually d' filters whose sizes are $k \times k \times c$. These filters produce a d'-dimensional feature map. On this feature map, the d-by-d' matrix P can be implemented as d filters whose sizes are $1 \times 1 \times d'$. So P corresponds to a convolutional layer with a 1×1 spatial support, which maps the d'-dimensional feature map to a d-dimensional one. The usage of 1×1 spatial filters to adjust dimensions has been adopted for designing network architectures [13, 19]. But in those papers, the 1×1 filters are used to reduce dimensions, while in our case they restore dimensions.

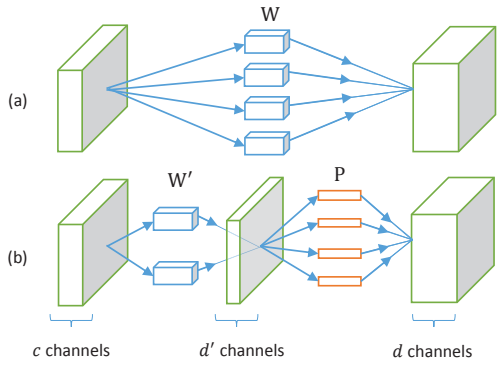

Figure 1. Illustration of the approximation. (a) An original layer with complexity $O(dk^2c)$. (b) An approximated layer with complexity reduced to $O(d'k^2c) + O(dd')$.

Note that the decomposition of $\mathrm{M} = \mathrm{PQ}^\top$ can be arbitrary. It does not impact the value of $\mathbf{y}$ computed in Eqn.(3). A simple decomposition is the Singular Vector Decomposition (SVD) [5]: $\mathrm{M} = \mathrm{U}_{d'}\mathrm{S}_{d'}{\mathrm{V}_{d'}}^\top$, where $\mathrm{U}_{d'}$ and $\mathrm{V}_{d'}$ are d-by-d' column-orthogonal matrices and $\mathrm{S}_{d'}$ is a d'-by-d' diagonal matrix. Then we can obtain $\mathrm{P} = \mathrm{U}_{d'}\mathrm{S}_{d'}^{1/2}$ and $\mathrm{Q} = \mathrm{V}_{d'}\mathrm{S}_{d'}^{1/2}$.

In practice the low-rank assumption is an approximation, and the computation in Eqn.(3) is approximate. To find an approximate low-rank subspace, we optimize the following problem:

$$\min_{\mathrm{M}} \sum_i \|(\mathbf{y}_i - \bar{\mathbf{y}}) - \mathrm{M}(\mathbf{y}_i - \bar{\mathbf{y}})\|_2^2, \tag{4}$$
$$s.t. \quad rank(\mathrm{M}) \leq d'.$$

Here $\mathbf{y}_i$ is a response sampled from the feature maps in the training set. This problem can be solved by SVD [5] or actually Principal Component Analysis (PCA): let Y be the d-by-n matrix concatenating n responses with the mean subtracted, compute the eigen-decomposition of the covariance matrix $\mathrm{YY}^\top = \mathrm{USU}^\top$ where U is an orthogonal matrix and S is diagonal, and $\mathrm{M} = \mathrm{U}_{d'}{\mathrm{U}_{d'}}^\top$ where $\mathrm{U}_{d'}$ are the first d' eigenvectors. With the matrix M computed, we can find $\mathrm{P} = \mathrm{Q} = \mathrm{U}_{d'}$.

How good is the low-rank assumption of the responses? We sample the responses from a CNN model (with 7 convolutional layers, detailed in Sec. 3) trained on ImageNet [2]. For the responses of a convolutional layer (from 3,000 randomly sampled training images), we compute the eigenvalues of their covariance matrix and then plot the sum of the largest eigenvalues (Fig. 2). We see that substantial energy is in a small portion of the largest eigenvectors. For example, in the Conv2 layer ($d = 256$) the first 128 eigenvectors contribute over 99.9% energy; in the Conv7 layer

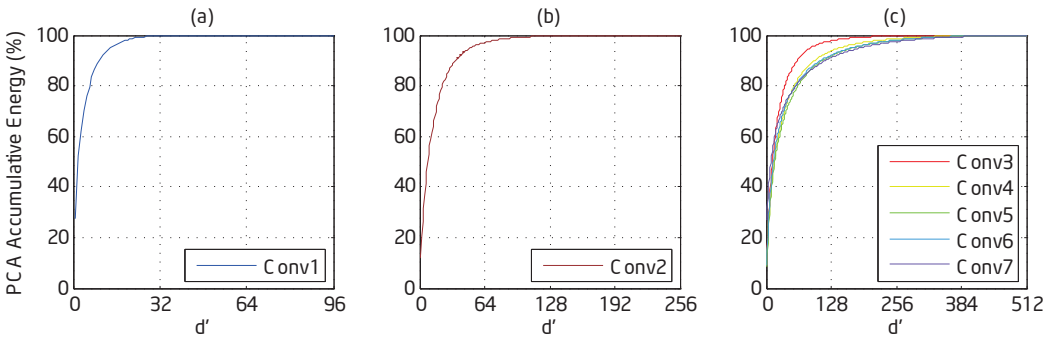

Figure 2. PCA accumulative energy of the responses in each layer, presented as the sum of largest d' eigenvalues (relative to the total energy when $d' = d$). Here the filter number d is 96 for Conv1, 256 for Conv2, and 512 for Conv3-7 (detailed in Table 1).

($d = 512$), the first 256 eigenvectors contribute over 95% energy. This indicates that we can use a fraction of the filters to precisely approximate the original filters.

The low-rank behavior of the responses $\mathbf{y}$ is because of the low-rank behaviors of the filters W and the inputs $\mathbf{x}$. While the low-rank assumptions of filters have been adopted in recent work [3, 10], we further adopt the low-rank assumptions of the filter input $\mathbf{x}$, which is a local volume and should have correlations. The responses $\mathbf{y}$ will have lower rank than W and x, so the approximation can be more precise. In our optimization (4), we directly address the low-rank subspace of $\mathbf{y}$.

2.2. The Nonlinear Case

Next we investigate the case of using nonlinear units. We use $r(\cdot)$ to denote the nonlinear operator. In this paper we focus on the Rectified Linear Unit (ReLU) [14]: $r(\cdot) = \max(\cdot, 0)$. A nonlinear response is given by $r(\mathrm{W}\mathbf{x})$ or simply $r(\mathbf{y})$. We minimize the reconstruction error of the nonlinear responses:

$$\min_{\mathrm{M},\mathbf{b}} \sum_i \|r(\mathbf{y}_i) - r(\mathrm{M}\mathbf{y}_i + \mathbf{b})\|_2^2, \quad s.t. \quad rank(\mathrm{M}) \leq d'. \tag{5}$$

Here $\mathbf{b}$ is a new bias to be optimized, and $r(\mathrm{M}\mathbf{y} + \mathbf{b}) = r(\mathrm{MW}\mathbf{x} + \mathbf{b})$ is the nonlinear response computed by the approximated filters.

The above problem is challenging due to the nonlinearity and the low-rank constraint. To find a feasible solution, we relax it as:

$$\min_{\mathrm{M},\mathbf{b},\{\mathbf{z}_i\}} \sum_i \|r(\mathbf{y}_i) - r(\mathbf{z}_i)\|_2^2 + \lambda\|\mathbf{z}_i - (\mathrm{M}\mathbf{y}_i + \mathbf{b})\|_2^2 \quad s.t. \quad rank(\mathrm{M}) \leq d'. \tag{6}$$

Here $\{\mathbf{z}_i\}$ is a set of auxiliary variables of the same size as $\{\mathbf{y}_i\}$. λ is a penalty parameter. If $\lambda \to \infty$, the solution to (6) will converge to the solution to (5) [23]. We adopt an alternating solver, fixing $\{\mathbf{z}_i\}$ and solving for M, $\mathbf{b}$ and vice versa.

(i) The subproblem of M, b. In this case, $\{\mathbf{z}_i\}$ are fixed. It is easy to show $\mathbf{b} = \bar{\mathbf{z}} - \mathrm{M}\bar{\mathbf{y}}$ where $\bar{\mathbf{z}}$ is the sample mean of $\{\mathbf{z}_i\}$. Substituting $\mathbf{b}$ into the objective function, we obtain the problem involving M:

$$\min_{\mathrm{M}} \sum_i \|(\mathbf{z}_i - \bar{\mathbf{z}}) - \mathrm{M}(\mathbf{y}_i - \bar{\mathbf{y}})\|_2^2, \quad s.t. \quad rank(\mathrm{M}) \leq d'. \tag{7}$$

Let Z be the d-by-n matrix concatenating the vectors of $\{\mathbf{z}_i - \bar{\mathbf{z}}\}$. We rewrite the above problem as:

$$\min_{\mathrm{M}} \|\mathrm{Z} - \mathrm{MY}\|_{\mathrm{F}}^2, \quad s.t. \quad rank(\mathrm{M}) \leq d'. \tag{8}$$

Here $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. This optimization problem is a Reduced Rank Regression problem [6, 21, 20], and it can be solved by a kind of Generalized Singular Vector Decomposition (GSVD) [6, 21, 20]. The solution is as follows. Let $\hat{\mathrm{M}} = \mathrm{ZY}^\top(\mathrm{YY}^\top)^{-1}$. The GSVD is applied on $\hat{\mathrm{M}}$ as $\hat{\mathrm{M}} = \mathrm{USV}^\top$, such that U is a d-by-d orthogonal matrix satisfying $\mathrm{U}^\top\mathrm{U} = \mathrm{I}_d$ where I_d is a d-by-d identity matrix, and V is a d-by-d matrix satisfying $\mathrm{V}^\top\mathrm{YY}^\top\mathrm{V} = \mathrm{I_d}$ (called *generalized orthogonality*). Then the solution M to (8) is given by $\mathrm{M} = \mathrm{U}_{d'}\mathrm{S}_{d'}{\mathrm{V}_{d'}}^\top$ where $\mathrm{U}_{d'}$ and $\mathrm{V}_{d'}$ are the first d' columns of U and V and $\mathrm{S}_{d'}$ are the largest d' singular values. We can further show that if $\mathrm{Z} = \mathrm{Y}$ (so the problem in (7) becomes (4)), this solution degrades to computing the eigen-decomposition of $\mathrm{YY}^\top$.

(ii) The subproblem of $\{\mathbf{z}_i\}$. In this case, M and $\mathbf{b}$ are fixed. Then in this subproblem each element z_{ij} of each vector $\mathbf{z}_i$ is independent of any other. So we solve a 1-dimensional optimization problem as follows:

$$\min_{z_{ij}} (r(y_{ij}) - r(z_{ij}))^2 + \lambda(z_{ij} - y'_{ij})^2, \tag{9}$$

where y'_{ij} is the j-th entry of $\mathrm{M}\mathbf{y}_i + \mathbf{b}$. We can separately consider $z_{ij} \geq 0$ and $z_{ij} < 0$ and remove the ReLU operator. Then we can derive the solution as follows: let

$$z'_{ij} = \min(0, y'_{ij}) \tag{10}$$

$$z''_{ij} = \max(0, \frac{\lambda \cdot y'_{ij} + r(y_{ij})}{\lambda + 1}) \tag{11}$$

then $z_{ij} = z'_{ij}$ if z'_{ij} gives a smaller value in (9) than z''_{ij}, and otherwise $z_{ij} = z''_{ij}$.

Although we focus on the ReLU, our method is applicable for other types of nonlinearities. The subproblem in (9) is a 1-dimensional nonlinear least squares problem, so can be solved by gradient descent or simply line search. We plan to study this issue in the future.

We alternatively solve (i) and (ii). The initialization is given by the solution to the linear case (4). We warm up the solver by setting the penalty parameter $\lambda = 0.01$ and run 25 iterations. Then we increase the value of λ. In theory, λ should be gradually increased to infinity [23]. But we find that it is difficult for the iterative solver to make progress if λ is too large. So we increase λ to 1, run 25 more iterations, and use the resulting M as our solution. Then we compute P and Q by SVD on M.

2.3. Asymmetric Reconstruction for Multi-Layer

To accelerate a whole network, we apply the above method sequentially on each layer, from the shallow layers to the deeper ones. If a previous layer is approximated, its error can be accumulated when the next layer is approximated. We propose an asymmetric reconstruction method to address this issue.

Let us consider a layer whose input feature map is not precise due to the approximation of the previous layer/layers. We denote the approximate input to the current layer as $\hat{\mathbf{x}}$. For the training samples, we can still compute its non-approximate responses as $\mathbf{y} = \mathrm{W}\mathbf{x}$. So we can optimize an "asymmetric" version of (5):

$$\min_{\mathrm{M},\mathbf{b}} \sum_i \|r(\mathrm{W}\mathbf{x}_i) - r(\mathrm{M}\mathrm{W}\hat{\mathbf{x}}_i + \mathbf{b})\|_2^2, \tag{12}$$

$$s.t. \quad rank(\mathrm{M}) \leq d'.$$

Here in the first term $\mathbf{x}_i$ is the non-approximate input, while in the second term $\hat{\mathbf{x}}_i$ is the approximate input due to the previous layer. We need not use $\hat{\mathbf{x}}_i$ in the first term, because $r(\mathrm{W}\mathbf{x}_i)$ is the real outcome of the original network and thus is more precise. On the other hand, we do not use $\mathbf{x}_i$ in the second term, because $r(\mathrm{M}\mathrm{W}\hat{\mathbf{x}}_i + \mathbf{b})$ is the actual operation of the approximated layer. This asymmetric version can reduce the accumulative errors when multiple layers are approximated. The optimization problem in (12) can be solved using the same algorithm as for (5).

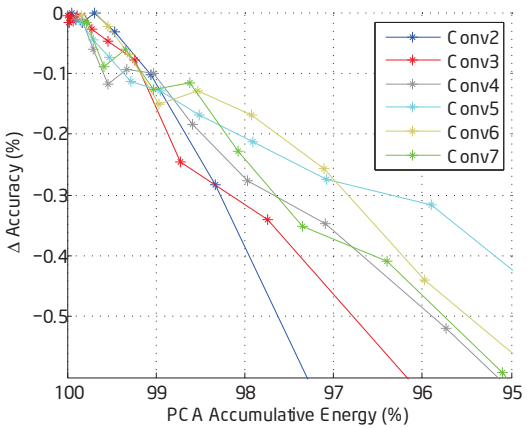

Figure 3. PCA accumulative energy and the accuracy rates (top-5). Here the accuracy is evaluated using the linear solution (the nonlinear solution has a similar trend). Each layer is evaluated independently, with other layers not approximated. The accuracy is shown as the difference to no approximation.

2.4. Rank Selection for Whole-Model Acceleration

In the above, the optimization is based on a target d' of each layer. d' is the only parameter that determines the complexity of an accelerated layer. But given a desired speedup ratio of the *whole model*, we need to determine the proper rank d' used for each layer.

Our strategy is based on an empirical observation that the PCA energy is related to the classification accuracy after approximations. To verify this observation, in Fig. 3 we show the classification accuracy (represented as the difference to no approximation) *vs*. the PCA energy. Each point in this figure is empirically evaluated using a value of d'. 100% energy means no approximation and thus no degradation of classification accuracy. Fig. 3 shows that the classification accuracy is roughly linear on the PCA energy.

To simultaneously determine the rank for each layer, we further assume that the whole-model classification accuracy is roughly related to the product of the PCA energy of all layers. More formally, we consider this objective function:

$$\mathcal{E} = \prod_l \sum_{a=1}^{d'_l} \sigma_{l,a} \tag{13}$$

Here $\sigma_{l,a}$ is the a-th largest eigenvalue of the layer l, and $\sum_{a=1}^{d'_l} \sigma_{l,a}$ is the PCA energy of the largest d'_l eigenvalues in the layer l. The product $\prod_l$ is over all layers to be approximated. The objective $\mathcal{E}$ is assumed to be related to the accuracy of the approximated whole network. Then we optimize this problem:

$$\max_{\{d'_l\}} \mathcal{E}, \qquad s.t. \quad \sum_l \frac{d'_l}{d_l} C_l \leq C. \tag{14}$$

layer	filter size	# channels	# filters	stride	output size	complexity (%)	# of zeros
Conv1	7×7	3	96	2	109×109	3.8	0.49
Pool1	3×3			3	37×37		
Conv2	5×5	96	256	1	35×35	17.3	0.62
Pool2	2×2			2	18×18		
Conv3	3×3	256	512	1	18×18	8.8	0.60
Conv4	3×3	512	512	1	18×18	17.5	0.69
Conv5	3×3	512	512	1	18×18	17.5	0.69
Conv6	3×3	512	512	1	18×18	17.5	0.68
Conv7	3×3	512	512	1	18×18	17.5	0.95

Table 1. The architecture of the model. Each convolutional layer is followed by ReLU. The final convolutional layer is followed by a spatial pyramid pooling layer [8] that have 4 levels ($\{6 \times 6, 3 \times 3, 2 \times 2, 1 \times 1\}$, totally 50 bins). The resulting 50×512-d is fed into the 4096-d fc layer (fc6), followed by another 4096-d fc layer (fc7) and a 1000-way softmax layer. The convolutional complexity is the theoretical time complexity, shown as relative numbers to the total convolutional complexity. The (relative) number of zeros is the calculated on the responses of the layer, which shows the "sparsity" of the layer.

Here d_l is the original number of filters in the layer l, and C_l is the original time complexity of the layer l. So $\frac{d'_l}{d_l} C_l$ is the complexity after the approximation. C is the total complexity after the approximation, which is given by the desired speedup ratio. This problem means that we want to maximize the accumulated accuracy subject to the time complexity constraint.

The problem in (14) is a combinatorial problem [15]. So we adopt a greedy strategy to solve it. We initialize d'_l as d_l, and consider the set $\{\sigma_{l,a}\}$. In each step we remove an eigenvalue σ_{l,d'_l} from this set, chosen from a certain layer l. The relative reduction of the objective is $\triangle\mathcal{E}/\mathcal{E} = \sigma_{l,d'} / \sum_{a=1}^{d'_l} \sigma_{l,a}$, and the reduction of complexity is $\triangle C = \frac{1}{d_l} C_l$. Then we define a measure as $\frac{\triangle\mathcal{E}/\mathcal{E}}{\triangle C}$. The eigenvalue σ_{l,d'_l} that has the smallest value of this measure is removed. Intuitively, this measure favors a small reduction of $\triangle\mathcal{E}/\mathcal{E}$ and a large reduction of complexity $\triangle C$. This step is greedily iterated, until the constraint of the total complexity is achieved.

2.5. Discussion

In our formulation, we focus on reducing the number of filters (from d to d'). There are algorithmic advantages of operating on the "d" dimension. Firstly, this dimension can be easily controlled by the rank constraint $rank(\mathrm{M}) \leq d'$. This constraint enables closed-form solutions, *e.g.*, PCA to the problem (4) or GSVD to the subproblem (7). Secondly, the optimized low-rank projection M can be exactly decomposed into low-dimensional filters (P and Q) by SVD. These simple and close-form solutions can produce good results using a very small subset of training images (3,000 out of one million).

3. Experiments

We evaluate on the "SPPnet (Overfeat-7)" model [8], which is one of the state-of-the-art models for ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 [16]. This model (detailed in Table 1) has a similar architecture to the Overfeat model [17], but has 7 convolutional layers. A spatial pyramid pooling layer [8] is used after the last convolutional layer, which improves the classification accuracy. We train the model on the 1000-class dataset of ImageNet 2012 [2, 16], following the details in [8].

We evaluate the "top-5 error" (or simply termed as "error") using single-view testing. The view is the center 224×224 region cropped from the resized image whose shorter side is 256. The single-view error rate of the model is 12.51% on the ImageNet validation set, and the increased error rates of the approximated models are all based on this number. For completeness, we report that this model has 11.1% error using 10-view test and 9.3% using 98-view test.

We use this model due to the following reasons. First, its architecture is similar to many existing models [11, 24, 17, 1] (such as the first/second layers and the cascade usage of 3×3 filters), so we believe most observations should be valid on other models. Second, on the other hand, this model is deep (7-conv.) and the computation is more uniformly distributed among the layers (see "complexity" in Table 1). A similar behavior exhibits on the compelling VGG-16/19 models [18]. The uniformly distributed computation indicates that most layers should be accelerated for an overall speedup.

For the training of the approximations as in (4), (6), and (12), we randomly sample 3,000 images from the ImageNet training set and use their responses as the training samples.

3.1. Single-Layer: Linear *vs*. Nonlinear

In this subsection we evaluate the single-layer performance. When evaluating a single approximated layer, the rest layers are unchanged and not approximated. The speedup ratio (involving that single layer only) is shown as the theoretical ratio computed by the complexity.

In Fig. 4 we compare the performance of our linear so-

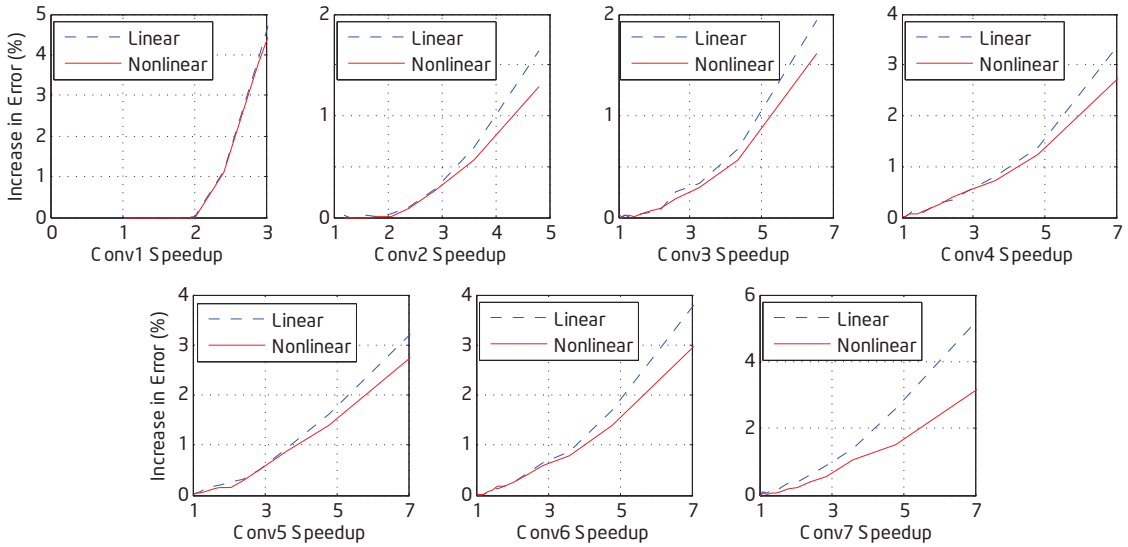

Figure 4. **Linear vs. Nonlinear**: single-layer performance of accelerating Conv1 to Conv7. The speedup ratios are computed by the theoretical complexity, but is nearly the same as the actual speedup ratios in our CPU/GPU implementation. The error rates are top-5 single-view, and shown as the increase of error rates compared with no approximation (*smaller is better*).

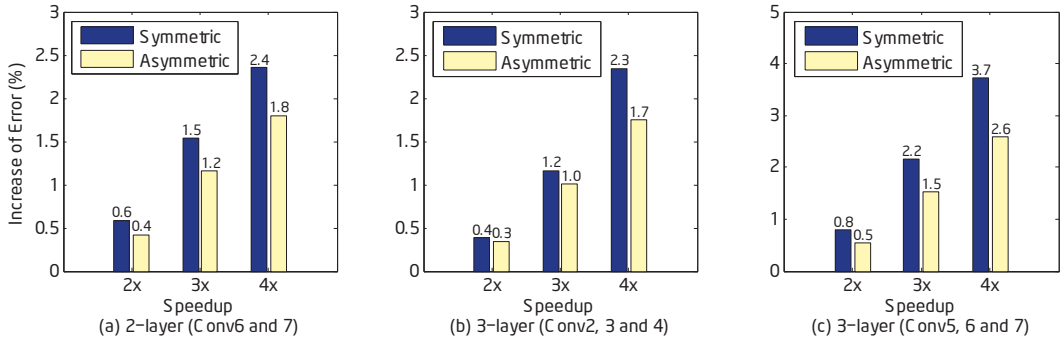

Figure 5. **Symmetric vs. Asymmetric**: the cases of 2-layer and 3-layer approximation. The speedup is computed by the complexity of the layers approximated. (a) Approximation of Conv6 & 7. (b) Approximation of Conv2, 3 & 4. (c) Approximation of Conv5, 6 & 7.

lution (4) and nonlinear solution (6). The performance is displayed as *increase of error rates* (decrease of accuracy) *vs*. the speedup ratio of that layer. Fig. 4 shows that the nonlinear solution consistently performs better than the linear solution. In Table 1, we show the sparsity (the portion of zero activations after ReLU) of each layer. A zero activation is due to the truncation of ReLU. The sparsity is over 60% for Conv2-7, indicating that the ReLU takes effect on a substantial portion of activations. This explains the discrepancy between the linear and nonlinear solutions. Especially, the Conv7 layer has a sparsity of 95%, so the advantage of the nonlinear solution is more obvious.

Fig. 4 also shows that when accelerating only a single layer by $2\times$, the increased error rates of our solutions are rather marginal or ignorable. For the Conv2 layer, the error rate is increased by $< 0.1\%$; for the Conv3-7 layers, the error rate is increased by $< 0.2\%$.

We also notice that for Conv1, the degradation is ignorable on or below $2\times$ speedup ($1.8\times$ corresponds to $d' = 32$). This can be explained by Fig. 2(a): the PCA energy has almost no loss when $d' \geq 32$. But the degradation can grow quickly for larger speedup ratios, because in this layer the channel number $c = 3$ is small and d' needs to be reduced drastically to achieve the speedup ratio. So in the following, we will use $d' = 32$ for Conv1.

3.2. Multi-Layer: Symmetric *vs*. Asymmetric

Next we evaluate the performance of asymmetric reconstruction as in the problem (12). We demonstrate approximating 2 layers or 3 layers. In the case of 2 layers, we show the results of approximating Conv6 and 7; and in the case of 3 layers, we show the results of approximating Conv5-7

speedup	rank sel.	Conv1	Conv2	Conv3	Conv4	Conv5	Conv6	Conv7	err. ↑ %
2×	no	32	110	199	219	219	219	219	1.18
2×	**yes**	32	83	182	211	239	237	253	**0.93**
2.4×	no	32	96	174	191	191	191	191	1.77
2.4×	**yes**	32	74	162	187	207	205	219	**1.35**
3×	no	32	77	139	153	153	153	153	2.56
3×	**yes**	32	62	138	149	166	162	167	**2.34**
4×	no	32	57	104	115	115	115	115	4.32
4×	**yes**	32	50	112	114	122	117	119	**4.20**
5×	no	32	46	83	92	92	92	92	6.53
5×	**yes**	32	41	94	93	98	92	90	**6.47**

Table 2. **Whole-model acceleration with/without rank selection**. The speedup ratios shown here involve all convolutional layers (Conv1-Conv7). We fix $d' = 32$ in Conv1. In the case of no rank selection, the speedup ratio of each other layer is the same. The solver is the asymmetric version. Each column of Conv1-7 shows the rank d' used, which is the number of filters after approximation. The error rates are top-5 single-view, and shown as the increase of error rates compared with no approximation (*smaller is better*).

or Conv2-4. The comparisons are consistently observed for other cases of multi-layer.

We sequentially approximate the layers involved, from a shallower one to a deeper one. In the asymmetric version (12), $\hat{\mathbf{x}}$ is from the output of the previous approximated layer (if any), and $\mathbf{x}$ is from the output of the previous non-approximate layer. In the symmetric version (5), the response $\mathbf{y} = \mathrm{M}\mathbf{x}$ where $\mathbf{x}$ is from the output of the previous non-approximate layer. We have also tried another symmetric version of $\mathbf{y} = \mathrm{M}\hat{\mathbf{x}}$ where $\hat{\mathbf{x}}$ is from the output of the previous approximated layer (if any), and found this symmetric version is even worse.

Fig. 5 shows the comparisons between the symmetric and asymmetric versions. The asymmetric solution has significant improvement over the symmetric solution. For example, when only 3 layers are approximated simultaneously (like Fig. 5 (c)), the improvement is over 1.0% when the speedup is 4×. This indicates that the accumulative error rate due to multi-layer approximation can be effectively reduced by the asymmetric version.

When more and all layers are approximated simultaneously (as below), if without the asymmetric solution, the error rates will increase more drastically.

3.3. Whole-Model: with/without Rank Selection

In Table 2 we show the results of whole-model acceleration. The solver is the asymmetric version. For Conv1, we fix $d' = 32$. For other layers, when the rank selection is not used, we adopt the same speedup ratio on each layer and determine its desired rank d' accordingly. When the rank selection is used, we apply it to select d' for Conv2-7. Table 2 shows that the rank selection consistently outperforms the counterpart without rank selection. The advantage of rank selection is observed in both linear and nonlinear solutions.

In Table 2 we notice that the rank selection often chooses a higher rank d' (than the no rank selection) in Conv5-7. For example, when the speedup is 3×, the rank selection assigns $d' = 167$ to Conv7, while this layer only requires $d' = 153$ to achieve 3× single-layer speedup of itself. This can be explained by Fig. 2(c). The energy of Conv5-7 is less concentrated, so these layers require higher ranks to achieve good approximations.

3.4. Comparisons with Previous Work

We compare with Jaderberg *et al*.'s method [10], which is a recent state-of-the-art solution to efficient evaluation. This method mainly operates on the spatial domain. It decomposes a $k \times k$ spatial support into a cascade of $k \times 1$ and $1 \times k$ spatial supports. This method focuses on the linear reconstruction error. The SGD solver is adopted for optimization. In the paper of [10], their method is only evaluated on a single layer of a model trained for ImageNet.

Our comparisons are based on our re-implementation of [10]. We use the *Scheme 2* decomposition in [10] and its filter reconstruction version, which is the one used for ImageNet as in [10]. Our re-implementation of [10] gives a 2× single-layer speedup on Conv2 and $< 0.2\%$ increase of error. As a comparison, in [10] it reports 0.5% increase of error on Conv2 under a 2× single-layer speedup, evaluated on another Overfeat model [17]. For whole-model speedup, we adopt this method sequentially on Conv2-7 using the same speedup ratio. We do not apply this method on Conv1, because this layer has a small fraction of complexity while the spatial decomposition leads to considerable error on this layer if using a speedup ratio similar to other layers.

In Fig. 6 we compare our method with Jaderberg *et al*.'s [10] for whole-model speedup. The speedup ratios are the theoretical complexity ratios involving all convolutional layers. Our method is the asymmetric version and with rank selection (denoted as "*our asymmetric*"). Fig. 6 shows that when the speedup ratios are large (4× and 5×), our method outperforms Jaderberg *et al*.'s method significantly. For ex-

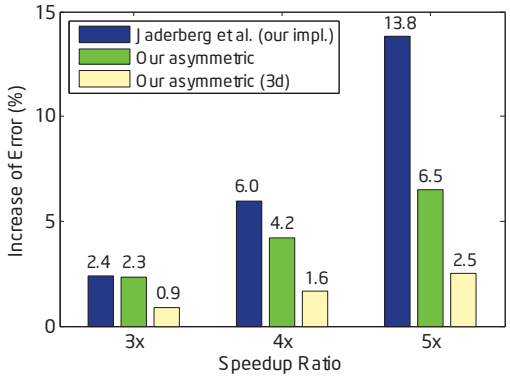

Figure 6. Comparisons with Jaderberg *et al*.'s spatial decomposition method [10]. The error rates are top-5 single-view, and shown as **the increase of error rates** compared with no approximation (*smaller is better*).

ample, when the speedup ratio is $4\times$, the increased error rate of our method is 4.2%, while Jaderberg *et al*.'s is 6.0%. Jaderberg *et al*.'s result degrades quickly when the speedup ratio is getting large, while ours degrades more slowly. This is indicates the effects of our method for reducing accumulative error. In our CPU implementation, both methods have similar actual speedup ratio for a given theoretical speedup, for example, $3.55\times$ actual for $4\times$ theoretical speedup. It is because the overhead for both methods mainly comes from fully-connected layers and other layers.

Because our asymmetric solution can effectively reduce the accumulated error, we can approximate a layer by the two methods simultaneously, and the asymmetric reconstruction of the next layer can reduce the error accumulated by the two methods. As discussed in Sec. 2.5, our method is based on the channel dimension (d), while Jaderberg *et al*.'s method mainly exploits the decomposition of the two spatial dimensions. These two mechanisms are complementary, so we conduct the following sequential strategy. The Conv1 layer is approximated using our model only. Then for the Conv2 layer, we first apply our method. The approximated layer has d' filters whose sizes are $k \times k \times c$ followed by 1×1 filters (as in Fig. 1(b)). Next we apply Jaderberg *et al*.'s method to decompose the spatial support into a cascade of $k \times 1$ and $1 \times k$ filters (*Scheme 2* [10]). This gives a 3-dimensional approximation of Conv2. Then we apply our method on Conv3. Now the asymmetric solver will take the responses approximated by the two mechanisms as the input, while the reconstruction target is still the responses of the original network. So while Conv2 has been approximated twice, the asymmetric solver of Conv3 can partially reduce the accumulated error. This process is sequentially adopted in the layers that follow.

In Fig. 6 we show the results of this 3-dimensional decomposition strategy (denoted as "*our asymmetric (3d)*").

model	speedup solution	top-5 err. (1-view)	top-5 err. (10-view)	CPU (ms)	GPU (ms)
AlexNet [11]	-	18.8	16.0	273	2.37
SPPnet, $4\times$ (Overfeat-7)	[10]	18.5	15.6	278	2.41
	our asym.	**16.7**	**14.4**	271	2.62
	our asym. (3d)	**14.1**	**12.0**	267	2.32

Table 3. Comparisons of network performance. The top-5 error is absolute values (not the increased number). The running time is per view on a CPU (single thread, with SSE) or a GPU.

We set the speedup ratios of both mechanisms to be equal: *e.g*., if the speedup ratio of the whole model is $r\times$, then we use $\sqrt{r}\times$ for both. Fig. 6 shows that this strategy leads to significantly smaller increase of error. For example, when the speedup is $5\times$, the error is increased by only 2.5%. This is because the speedup ratio is accounted by all three dimensions, and the reduction of each dimension is lower. Our asymmetric solver effectively controls the accumulative error even if the multiple layers are decomposed extensively.

Finally, we compare the accelerated whole model with the well-known "*AlexNet*" [11]. The comparison is based on our re-implementation of AlexNet. The architecture is the same as in [11] except that the GPU splitting is ignored. Besides the standard strategies used in [11], we train this model using the 224×224 views cropped from resized images whose shorter edge is 256 [9]. Our re-implementation of this model has top-5 single-view error rate as 18.8% (10-view top-5 16.0% and top-1 37.6%). This is better than the one reported in [11][1].

Table 3 shows the comparisons on the accelerated models and AlexNet. The error rates in this table are the absolute value (not the increased number). The time is the actual running time per view, on a C++ implementation and Intel i7 CPU (2.9GHz). The model accelerated by our asymmetric solver (channel-only) has 16.7% error, and by our asymmetric solver (3d) has 14.1% error. This means that the accelerated model is 4.7% more accurate than AlexNet, while its speed is nearly the same as AlexNet. As a common practice [11], we also evaluate the 10-view score of the models. Our accelerated model achieves 12.0% error, which means only **0.9%** increase of error with $4\times$ speedup.

We also evaluate our approximation method on the scene *character classification* model released by [10]. Our asymmetric (3d) solution achieves $4.5\times$ speedup with only a drop of 0.7% in classification accuracy, which is better than the 1% drop for the same speedup reported by [10].

Acknowledgements. This work is supported in part by the National Nature Science Foundation of China Grant No. 61303121, and Microsoft Research Asia Collaborative Research Award.

[1] In [11] the 10-view error is top-5 18.2% and top-1 40.7%.

References

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[3] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[5] G. H. Golub and C. F. van Van Loan. Matrix computations. 1996.

[6] J. C. Gower and G. B. Dijksterhuis. *Procrustes problems*, volume 3. Oxford University Press Oxford, 2004.

[7] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*, pages 297–312, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *arXiv:1406.4729v2*, 2014.

[9] A. G. Howard. Some improvements on deep convolutional neural network based image classification. In *arXiv:1312.5402*, 2013.

[10] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

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

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[13] M. Lin, Q. Chen, and S. Yan. Network in network. In *arXiv:1312.4400*, 2013.

[14] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[15] C. R. Reeves. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

[16] 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. *arXiv:1409.0575*, 2014.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2014.

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

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

[20] Y. Takane and H. Hwang. Regularized linear and kernel redundancy analysis. *Computational Statistics & Data Analysis*, pages 394–405, 2007.

[21] Y. Takane and S. Jung. Generalized constrained redundancy analysis. *Behaviormetrika*, pages 179–192, 2006.

[22] 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*, 2011.

[23] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *CVPR*, 2010.

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