Low-rank Compression of Neural Nets: Learning the Rank of Each Layer

Yerlan Idelbayev
Dept. CSE, University of California, Merced
yidelbayev@ucmerced.edu

Miguel Á. Carreira-Perpiñán
Dept. CSE, University of California, Merced
mcarreira-perpinan@ucmerced.edu

Abstract

Neural net compression can be achieved by approximating each layer's weight matrix by a low-rank matrix. The real difficulty in doing this is not in training the resulting neural net (made up of one low-rank matrix per layer), but in determining what the optimal rank of each layer is—effectively, an architecture search problem with one hyperparameter per layer. We show that, with a suitable formulation, this problem is amenable to a mixed discrete-continuous optimization jointly over the ranks and over the matrix elements, and give a corresponding algorithm. We show that this indeed can select ranks much better than existing approaches, making low-rank compression much more attractive than previously thought. For example, we can make a VGG network faster than a ResNet and with nearly the same classification error.

Compressing large deep neural nets so that they can be deployed on resource-constrained devices, such as smart cameras or mobile phones, is a problem of considerable interest at present. Of the many existing compression forms we focus on low-rank compression, whose roots lie in matrix algebra, and where we replace a matrix $\mathbf{W}$ with another one having lower rank (which can thus be written as the product $\mathbf{U}\mathbf{V}^T$ of two smaller matrices). That low-rank compression can be very effective is obvious from its many applications in numerical linear algebra, image and signal processing, model order reduction, machine learning and statistics, and other areas in science and engineering. Indeed it has been applied for deep net compression (see related work), although it generally has been found less effective than other compression techniques such as weight quantization or pruning. However, a low-rank representation has the advantage of fast inference (particularly on GPUs), since it uses dense matrices having a local, regular and parallelizable memory access pattern.

A fundamental problem in low-rank compression is the selection of the rank. This is often not recognized because selecting the rank and the corresponding matrices $\mathbf{U}$ and $\mathbf{V}$ is very easy in some special but important cases, which accounts for the widespread use of low-rank compression. Indeed, say we are given a matrix $\mathbf{W}$ and want to find a low-rank matrix that produces the lowest approximation error to $\mathbf{W}$. The solution of this problem is known and is computationally very convenient: solving a single full singular value decomposition (SVD) we obtain the optimal solution for any desired rank and hence any desired approximation error [15]. We simply sort the singular values and pick as many as we need in increasing order to reach the desired approximation error. The matrices $\mathbf{U}$ and $\mathbf{V}$ are given by the corresponding singular vectors. *Note that we need not solve a separate optimization problem for each target value of the error or rank: we get all the possible solutions at once.* This property will prove useful in our algorithm later.

This property also holds in a special case of the model compression problem (well known in statistics as reduced-rank regression (RRR) [22, 41]): solving a linear regression fit but constraining the rank of the coefficient matrix to be at most r. This problem can be solved again for all possible values of the rank r by computing a single SVD of the data matrix (training input and output vectors).

Let us now see why this is much harder in the case of a deep net where we want to find both the rank for each layer matrix and the matrix coefficients so that some desired error is minimized and the ranks are constrained. The solution is not given anymore by computing a single SVD and examining a list of ranks and their errors. What is relatively easy (at least, doable in current practice with deep nets) is to find the weights of the matrix at each layer *if we fix ahead of time the rank r_k that we want to use in each layer k*. This is because we can then write the matrix at each layer as $\mathbf{U}_k\mathbf{V}_k^T$ and optimize this network as usual over all the $\mathbf{U}_k$ and $\mathbf{V}_k$ matrices. That is, we compute the gradient using the chain rule (easily done with automatic differentiation in deep learning frameworks such as PyTorch [39] or TensorFlow [1]) and use it with stochastic gradient descent (SGD) to converge (approximately at least) to a local minimizer.

This makes it obvious that the real problem is in determining optimal values for the set of ranks $r_1, \dots, r_K$. It also shows that the problem can be seen as a special case of architecture optimization, where we search both over architectures (i.e., the number of hidden units, or rank, within each layer) and over values of the matrices' weights. Hence, this is a hard, combinatorial problem which is exponential on the number of layers. Specifically, in a net with K layers of weights each having a maximum rank of R_k there are $\prod_{k=1}^{K} R_k$ combinations of rank choices, and each R_k can be thousands in large nets. Even having access to many GPUs over weeks of training makes it possible to train only a tiny subset of combinations. Practically, this means that one should try to guess reasonable values for these ranks and perhaps explore the space from that point a little by trying additional combinations, but this generally will provide solutions that may be far from optimal.

In this paper, we propose a way to find a good, approximate solution to this problem efficiently. We first formulate the problem in a way that incorporates our desire of minimizing the (say) classification loss of the net but with a model selection cost dependent on the ranks (memory or inference time, for example), and a rank constraint on each layer. We then show how this can be optimized over both ranks and weight values, by interleaving SGD steps that train the uncompressed net with SVD steps that determine the currently optimal rank and weight matrices. Our algorithm can be shown to explore architectures on the fly while monotonically decreasing a certain objective function which, in the limit, tends to a local solution of the problem (an approximate solution, since the problem is presumably NP-hard). We achieve this with a single run of the training algorithm, which is not much slower than training the uncompressed net itself, and therefore much faster than training multiple nets of different architectures. After reviewing related work (section 1), we formulate the problem (section 2), give our algorithm (section 3), and test it with neural nets having up to hundreds of layers (section 4).

1. Related work

Matrix factorizations, including low-rank factorization via the SVD and various forms of tensor factorization, have been extensively studied in theory and application [8, 9, 27, 46]. Likewise, neural net compression can be done with methods other than low-rank compression, such as weight quantization or pruning. In this section, we limit ourselves to works involving low-rank compression (of matrices or tensors) of deep neural nets in terms of model size or inference time.

Ranks assumed fixed Various types of low-rank decompositions of fully-connected and convolutional layers (with suitable reshaping) have been used [10, 11, 23, 32, 43, 45, 47, 48, 49, 52], as well as tensor decompositions such as CP [11, 30], Tucker [26] and Tensor-Train [14, 38]. Modifications and combinations of different decompositions are common too, e.g. constraining each filter in a convolutional layer to have unit rank [23] or to be a linear combination of filters of different rank [20, 21].

Algorithmically, early works used either data-dependent or data-independent methods to obtain a layerwise factorization of a trained, reference net, which determines the rank at each layer, and then fine-tuned the resulting net. Data-independent strategies minimize the distance $\left\|\mathbf{W} - \mathbf{U}\mathbf{V}^T\right\|$ between the original and compressed weight matrix, using a closed form solution such as the SVD [11, 38, 45, 47] or iterative optimization [23, 30]. Data-dependent strategies minimize either a data-dependent distance between the original and compressed matrix [11], or the distance of the output activations (at each layer) between the original and compressed matrix $\left\|\mathbf{W}\mathbf{x} - \mathbf{U}\mathbf{V}^T\mathbf{x}\right\|$, treating it as regression problem [10, 52].

Rank selection Some methods apply a heuristic to fix the ranks ahead of time, such as using a greedy approach or thresholding [25, 26, 47, 48, 52]. We use some of these (described in section 4.1) as baseline comparison in our experiments. Other work [47] proposes to train a neural net with a penalty that encourages (but does not actually constrain) the weight matrices to be low-rank. Since it does not handle the ranks directly, it requires a heuristic selection (thresholding) of the ranks after training. This thresholding can be alternated instead with the net training [48]. Finally, we are aware of only one method to jointly learn ranks and weights as part of the overall optimization [32]. This proposes a specific formulation to minimize the loss of a deep net with constraints on the total number of allowed weights and computation. This problem is solved approximately in an alternating manner, where one step optimizes over the weights via SGD, and another step optimizes over the ranks but requires the solution of a mixed-integer program (involving discrete and continuous variables, NP-hard) using the commercial software MOSEK. Our algorithm also has a step over the full-rank weight matrices and over the low-rank matrices and their ranks, but we show the latter can be solved exactly in closed form via an SVD per layer for both ranks and weights.

Another line of work is to use a convex relaxation of the rank, the nuclear norm (or trace norm) [13]. Minimizing the nuclear norm instead of the rank is easier and enjoys optimality guarantees under certain conditions in compressed sensing [3, 40]. The nuclear norm was used to reduce the number of parameters in single- [16] and multi-layer networks [2]. However, minimizing the nuclear norm is generally not equivalent to minimizing the rank with multi-layer networks, and it is not clear how cost functions such as FLOPs would relate to the nuclear norm. Our formulation has both advantages: it directly optimizes what we really want (a cost function of each integer rank) and does so exactly and efficiently in the step over the ranks.

2. Problem formulation

Assume we want to train a K-layer deep net, where layer k takes the form $\mathbf{z}_k = \sigma(\mathbf{W}_k \mathbf{z}_{k-1})$, $\sigma(\cdot)$ is an elementwise nonlinearity (e.g. ReLU), and $\mathbf{z}_k$ is the input at layer k (with $\mathbf{x} = \mathbf{z}_0$ and $\mathbf{y} = \mathbf{z}_K$ being the input and output of the net); we omit bias parameters to keep the notation simple. Let us denote the loss function (e.g. cross-entropy for classification on a training set) as $L(\mathbf{W})$ where $\mathbf{W} = (\mathbf{W}_1, \dots, \mathbf{W}_K)$ are the deep net weight matrices, where $\mathbf{W}_k$ is of dimension $a_k \times b_k$, and $b_k = a_{k-1}$ (so layer $k-1$ feeds into layer k). We then want to minimize the sum of this loss and a cost function C but constraining each rank not to exceed a maximum rank R_k at each layer. Hence we have:

$$\begin{aligned} \min_{\mathbf{W}} \ & L(\mathbf{W}) + \lambda\, C(\mathbf{W}) \\ \text{s.t.} \ & \operatorname{rank}(\mathbf{W}_k) \leq R_k,\ k = 1, \dots, K. \end{aligned} \tag{1}$$

Note that we must have $R_k \leq \min(a_k, b_k)$, which is the maximum rank possible at layer k. The constraint in (1) can be equivalently written as $\mathbf{W}_k = \mathbf{U}_k \mathbf{V}_k^T$ where $\mathbf{U}_k$ is of $a_k \times r_k$, $\mathbf{V}_k$ is of $b_k \times r_k$ and $r_k \in \{0, 1, \dots, R_k\}$, but r_k is an unknown parameter itself that we need to optimize over. In this equivalent formulation, we optimize over $\{\mathbf{W}_k, \mathbf{U}_k, \mathbf{V}_k, r_k\}_{k=1}^K$ jointly, which makes explicit the combinatorial nature of the problem.

We now define the cost function $C(\mathbf{W})$ as follows and make some key observations:

$$C(\mathbf{W}) = C(r_1, \dots, r_K) = C_1(r_1) + \dots + C_K(r_K). \tag{2}$$

A useful particular case is $C(\mathbf{W}) = \alpha_1 r_1 + \dots + \alpha_K r_K$ where $\alpha_1, \dots, \alpha_K \geq 0$ are constants. First, from an optimization point of view, C depends only on the ranks (not the actual weight values in $\mathbf{W}$) and is a separable function of them. With some manipulations in the next section, this will make it possible to solve a problem that has an apparently exponential cost over the number of layers K exactly in linear cost over K. The intuition is very similar to the "model selection on the fly" idea of [7].

Second, from a modeling point of view, C can represent several costs of interest in the context of neural net compression by appropriate choices of the coefficients α_k. For example, the memory occupied by $\mathbf{W}$ (in number of elements) results from $\alpha_k = a_k + b_k$ (since $\mathbf{W}_k$ is stored as $\mathbf{U}_k$ and $\mathbf{V}_k$, and the memory occupied by a floating-point number does not depend on its value). The runtime at inference to compute the output $\mathbf{y}$ of the net with an input $\mathbf{x}$ results (up to an additive constant) from those same coefficients. If convolutional layers are involved, the coefficients change according to which filters are reused but the memory and runtime remain a linear function of the ranks. Other costs, such as energy or bandwidth, can be written in this form, at least approximately. Hence, C plays the role of a model selection criterion. But unlike traditional criteria such as AIC, BIC or MDL [17], over which there is no general agreement, in our context there is a clear definition of cost (given by the system where the neural net will be deployed and the performance target required by the application, such as fast image classification in a camera of given memory size and CPU frequency). While this cost can be defined in different ways, it is not arbitrary. The hyperparameter $\lambda \geq 0$ naturally trades off the classification loss with the cost, and determines the distribution of ranks over the deep net layers—which varies in a complex way, as seen in our experiments.

3. Optimization algorithm

Let us rewrite (1) by introducing auxiliary variables $\mathbf{\Theta} = (\mathbf{\Theta}_1, \dots, \mathbf{\Theta}_K)$ and a constraint $\mathbf{\Theta} = \mathbf{W}$:

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{\Theta}, \mathbf{r}} \ & L(\mathbf{W}) + \lambda\, C(\mathbf{r}) \\ \text{s.t.} \ & \mathbf{W}_k = \mathbf{\Theta}_k,\ \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k,\ k = 1, \dots, K \end{aligned} \tag{3}$$

with $\mathbf{r} = (r_1, \dots, r_K)$. This formulation now takes the form of *model compression as constrained optimization* [4, 5, 6], for which a convenient "learning-compression (LC)" optimization algorithm can be generally applied, and which helps us capitalize on the separability of the cost function C. Following [4], we apply a penalty method and then alternating optimization. Below we give the algorithm for the quadratic-penalty method [37] for shortness, however, we implement the augmented Lagrangian version which works in a similar way but with the introduction of a Lagrange multiplier vector $\boldsymbol{\beta}$ of the same dimension as $\mathbf{W}$, see Alg. 1. After applying the quadratic-penalty we optimize the following while driving the penalty parameter $\mu \to \infty$ (norms are Frobenius):

$$\begin{aligned} Q(\mathbf{W}, \mathbf{\Theta}, \mathbf{r}; \mu) &= L(\mathbf{W}) + \lambda\, C(\mathbf{r}) + \frac{\mu}{2} \sum_{k=1}^K \|\mathbf{W}_k - \mathbf{\Theta}_k\|^2 \\ \text{s.t.} \quad & \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k,\ k = 1, \dots, K \end{aligned}$$

by using alternating optimization over $\mathbf{W}$ and $(\mathbf{\Theta}, \mathbf{r})$. The step over $\mathbf{W}$ ("learning (L)" step) has the form of a standard loss minimization $\min_{\mathbf{W}} L(\mathbf{W}) + \frac{\mu}{2} \sum_k \|\mathbf{W}_k - \mathbf{\Theta}_k\|^2$ but with a quadratic regularizer on $\mathbf{W}$ (since $\mathbf{\Theta}$ is fixed), and can be done using a standard algorithm to optimize the loss, e.g. SGD with deep nets. The step over $\mathbf{\Theta}$ and $\mathbf{r}$ ("compression (C)" step) separates into a problem for each layer's matrix $\mathbf{\Theta}_k$ and rank r_k, of the following form:

$$\begin{aligned} \min_{\mathbf{\Theta}_k, r_k} \ & \lambda\, C_k(r_k) + \frac{\mu}{2} \|\mathbf{W}_k - \mathbf{\Theta}_k\|^2 \\ \text{s.t.} \ & \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k. \end{aligned} \tag{4}$$

We recognize this problem as a standard low-rank approximation in the Frobenius norm with a penalty on the rank

Algorithm 1 Pseudocode (augmented Lagrangian version)

```
input training set, $K$-layer neural net with weights $\{\mathbf{W}_k\}$,
      hyperparameter $\lambda$, layerwise cost functions $\{C_k\}$
$\mathbf{W} = (\mathbf{W}_1, \dots, \mathbf{W}_K) \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W})$    reference net
$\mathbf{r} = (r_1, \dots, r_K) \leftarrow \mathbf{0}$    ranks
$\boldsymbol{\Theta} = (\boldsymbol{\Theta}_1, \dots, \boldsymbol{\Theta}_K) \leftarrow \mathbf{0}$    auxiliary weights
$\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_K) \leftarrow \mathbf{0}$    Lagrange multipliers
for $\mu = \mu_0 < \mu_1 < \dots < \infty$
  $\mathbf{W} \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W}) + \frac{\mu}{2}\|\mathbf{W} - \boldsymbol{\Theta} - \frac{1}{\mu}\boldsymbol{\beta}\|^2$    L step
  for $k = 1, \dots, K$    C step
    $\boldsymbol{\Theta}_k, r_k \leftarrow \arg\min_{\boldsymbol{\Theta}_k, r_k} \lambda\, C_k(r_k) + \frac{\mu}{2}\|\mathbf{W}_k - \boldsymbol{\Theta}_k - \frac{1}{\mu}\boldsymbol{\beta}_k\|^2$
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{W} - \boldsymbol{\Theta})$    multipliers step
  if $\|\mathbf{W} - \boldsymbol{\Theta}\|$ is small enough then exit the loop
return $\mathbf{W}, \boldsymbol{\Theta}, \mathbf{r}$
```

variable r_k (which is a scalar). This can be solved using the Eckhart-Young theorem [15, th. 2.4.8]. Assume w.l.o.g. $a_k \geq b_k$ and let $\mathbf{W}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^T$ be the SVD of $\mathbf{W}_k$, where $\mathbf{U}_k$ of $a_k \times b_k$ and $\mathbf{V}_k$ of $b_k \times b_k$ are orthogonal matrices, and $\mathbf{S}_k = \operatorname{diag}(s_1, \dots, s_{b_k})$ with $s_1 \geq \dots \geq s_{b_k} \geq 0$ (sorted singular values). Then problem (4) is equivalent to:

$$\min_r \lambda\, C_k(r) + \frac{\mu}{2} \sum_{i=r+1}^{R_k} s_{ki}^2 \quad \text{s.t.} \;\; r_k \in \{0, 1, \dots, R_k\} \quad (5)$$

which can be solved by enumeration, i.e., trying all $R_k + 1$ values of r. That is, the C step is solved exactly by 1) computing the full SVD of each $\mathbf{W}_k$ (although we only need its leading R_k singular values), 2) finding the optimal rank of each r_k from (5), and 3) forming each $\boldsymbol{\Theta}_k = \mathbf{U}_k(:, 1\!:\!r_k)\, \mathbf{S}_k(1\!:\!r_k, 1\!:\!r_k)\, \mathbf{V}(:, 1\!:\!r_k)^T$ based on the top r_k singular values and corresponding singular vectors. If $\lambda = 0$ this returns the usual solution $r_k = R_k$ where we pick the top R_k singular values, but with $\lambda > 0$ we get $r_k \leq R_k$ depending on the case. We can solve (5) by enumeration because a single SVD gives us the optimal singular vectors and singular values for all possible rank values, as mentioned in the introduction.

Overall, the LC algorithm operates by training the regularized model for a while with SGD over the full-rank matrices $\mathbf{W}_1, \dots, \mathbf{W}_K$ (with a regularization term given by each low-rank matrix $\boldsymbol{\Theta}_k$), and then obtaining each low-rank matrix $\boldsymbol{\Theta}_k$ with currently optimal rank r_k via a SVD of $\mathbf{W}_k$. The algorithm keeps two copies of each layer matrix: $\mathbf{W}_k$ of full (unrestricted) rank, and $\boldsymbol{\Theta}_k$ of rank $r_k \leq R_k$ determined within each C step. Both coincide in the limit $\mu \to \infty$. It is within the C step that the architecture changes at each step, effectively by doing a model selection over the rank of each matrix. Practically, rather than continuing to iterate L and C steps until convergence, at some iteration we fix the ranks, thereby fixing the architecture, and optimize it directly via SGD with the chain rule, which is faster.

4. Experiments

We evaluate our algorithm on multiple datasets and networks: LeNet300, LeNet5 on MNIST [31]; ResNets [18], VGG16 [44], and NIN [35] on CIFAR10 [28]; AlexNet [29] on ImageNet [42]; and compare our results to baselines and other relevant works. We choose both lean (ResNets) and large (AlexNet, VGG16) networks to demonstrate the power of rank-selection approach. Experiments are initialized from reasonably well-trained reference models with same or exceeding test accuracies reported in the literature.

We use our LC algorithm (augmented Lagrangian version) as follows throughout all experiments with minor changes (see suppl. mat. for details). In the L step we use Nesterov's accelerated gradient method [36] with momentum 0.9 on minibatches of size 128 (256 for MNIST) and decayed learning rate schedule of $\eta_0 \cdot a^m$ at the mth epoch. The initial learning rate η_0 is one of {0.0007,0.001}; the learning rate decay is one of {0.98,0.99}. We run each L step for 15 epochs (30 for MNIST). The C step requires, per layer, a SVD followed by a scalar rank selection. It runs for j iterations where $j \leq 60$ and uses a penalty parameter schedule $\mu_j = \mu_0 \cdot b^j$; we take μ_0 to be one of $\{5 \cdot 10^{-4}, 10^{-3}\}$ and $b \in \{1.2, 1.25\}$.

For each compressed model we report its training loss, test error, and ratio of storage (ρ_{storage}) and floating point operations (ρ_{FLOPs}). We calculate FLOPs based on the assumption of fused multiplication and addition, treating it as one FLOP. For example, a forward pass through a fully connected layer with weight matrix of $n \times m$ and bias of $n \times 1$ has nm multiplications and nm additions, for which we report nm FLOPs. See suppl. mat. for details.

For the cost function C of eq. (2) we use throughout the paper the deep net inference runtime (FLOPs added over all layers), which is a linear function of the layers' ranks. We report results for different values of the λ hyperparameter which trades off the classification loss L vs the cost C. Also, in one of our animations (suppl. mat.), we use storage (memory occupied by the parameters) instead. It is very interesting to observe how different the resulting rank distribution is over the layers. This is to be expected because, while memory and FLOPs are similar for fully-connected layers, they are very different in convolutional layers. Other suppl. mat. animations show how the ranks change over iterations of the LC algorithm, going up and down individually in a complex way and making large moves in rank space, and how this depends on λ and C.

Low-rank parametrization of convolutional layers A convolutional layer with n filters of c channels and $d \times d$ spatial resolution has ncd^2 parameters. We can parametrize it with one of the following low-rank structures:

Scheme 1 We can view the convolutional weights as a lin-

	$\lambda \times 10^{-3}$	rank per layer	FLOPs	# params.	$\log L$	E_{train}	E_{test}	ρ_{FLOPs}	ρ_{storage}
LeNet300	**R**	[300 100 10]	0.26M	0.26M	-3.68	0.00	1.98	1.00	1.00
	0.25	[35 16 9]	45 330	46 150	-4.11	0.00	1.87	5.87	5.77
	1	[24 10 9]	31 006	31 826	-4.02	0.00	2.06	8.59	8.36
	2	[18 9 9]	24 102	24 922	-3.88	0.00	2.39	11.04	10.68
LeNet5	**R**	[20 50 500 10]	2.29M	0.43M	-6.27	0.00	0.55	1.00	1.00
	0.4	[5 5 14 9]	328 390	26 855	-3.73	0.01	0.75	6.98	16.04
	1.0	[4 5 9 9]	295 970	20 310	-3.27	0.01	0.82	7.75	21.20
	1.1	[3 3 9 9]	199 650	19 165	-2.28	0.16	1.33	11.49	22.47

Table 1. Results of our algorithm with different λ values on LeNet300 and LeNet5 nets on MNIST (**R** = reference net). We report the rank selected for each layer, number and ratio of parameters and FLOPs of the compressed net, training loss L, and training and test error (%).

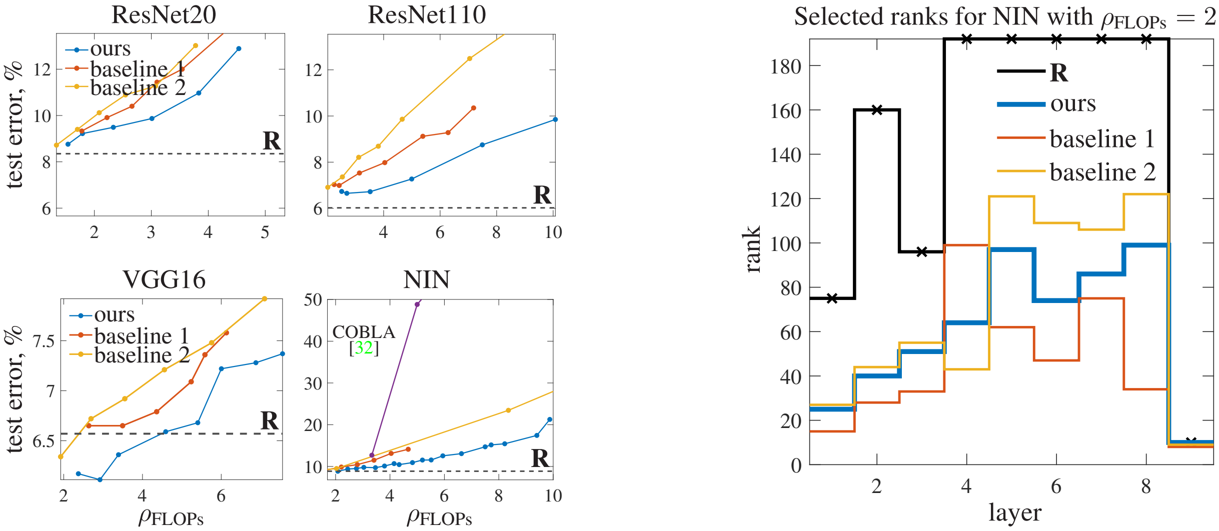

Figure 1. *Left*: our algorithm and baselines on CIFAR10 networks (ResNets, VGG16, NIN). We plot test error vs ρ_{FLOPs} ratio (reference net: horizontal dashed lines). *Right*: rank selected at each layer for a NIN, compressed to achieve $\rho_{\text{FLOPs}} = 2$ (reference: black × line).

ear layer with shape of $n \times cd^2$ applied to appropriately reshaped volumes of the input. The rank-r approximation then has two linear mappings with weight shapes of $n \times r$ and $r \times cd^2$, which can be efficiently implemented as a sequence of two convolutional layers, with r filters of shape $c \times d \times d$, and with n filters of shape $r \times 1 \times 1$ [32, 47, 48, and others].

Scheme 2 Alternatively, we can view the convolutional weights as a linear layer with shape of $nd \times cd$ applied to reshaped volumes of the input. For this scheme, an approximation of rank r will have two linear mappings with weight shapes of $nd \times r$ and $r \times cd$, which can be implemented as a sequence of two convolutional layers: first with r filters of $c \times d \times 1$ and second with n filters of $r \times 1 \times d$ [23, 45].

We use the same low-rank scheme throughout all layers in a network. We run experiments with scheme 1 on MNIST and CIFAR10 and with both schemes for ImageNet.

4.1. Comparison baselines to estimate the ranks

One simple, approximate way to apply low-rank compression to a deep net is to estimate the ranks with a heuristic so the architecture is determined and then training this as usual. We briefly describe two heuristics we use as baseline. In each case, first a reference net with full-rank weight matrices is trained, and then a lower rank is chosen for each layer as follows:

Baseline 1 [52] Pick ranks to maximize the accumulated sum of singular values over all the matrices subject to the resulting FLOPS of the net being within a proportion p of the full-rank net. They do this by a greedy algorithm, picking one rank at a time from one of the matrices.

Baseline 2 [47, 48] A simpler heuristic is to choose the lowest rank r_k of matrix k, separately for each layer k, such that its r_k-rank approximation error is within a $1-p$ proportion of the norm of the original matrix: $\left\|\mathbf{W}_k - \mathbf{U}_k\mathbf{V}_k^T\right\|_F \leq (1-p)\|\mathbf{W}_k\|_F$. They take p to be 0.95 [47] or $\{0.95, 0.99\}$ [48].

Essentially, this is the commonly used rule with PCA of picking the number of principal components that will preserve a target proportion $p \in [0, 1]$ of explained variance. It is done jointly over all layers in baseline 1, which is a hard optimization problem that is solved approximately, or layerwise in baseline 2, which can be done exactly. Having chosen a p value and estimated the ranks with a baseline, we retrain the new net using Nesterov SGD with learning rates to achieve as good performance as possible. We train for about twice the number of epochs required to train the reference networks, with a learning rate of 0.002 for ResNets and 0.001 for NIN and VGG16, decayed by 0.99 after every epoch. See suppl. mat. for details. Finally, we also compare with COBLA [32], which approximately optimizes the training loss over both ranks and weight matrices.

4.2. MNIST experiments

We train LeNet300 and LeNet5 networks on MNIST (60k grayscale images of 28×28 and 10 digit classes). We normalize the images to have pixels in $[0, 1]$ and subtract their mean. Table 1 shows the result of our algorithm with different values of λ (see suppl. mat. for full details and additional experiments). It is clear that significant compression can be achieved with no or little increase in error, and that the ranks found show a complex distribution over layers. We explore this further next.

4.3. CIFAR10 experiments and comparison

We train reference ResNets of different sizes (20, 32, 56 and 110 layers) following the procedure of the original paper [18]. NIN and VGG16 (adapted for CIFAR10) are trained using the same data augmentation as for ResNets. We compress these networks using the baselines described earlier and our algorithm (with various values of λ). For ResNets we compress the convolutional layers only, since the last, only fully-connected layer is very small (64×10).

Fig. 1 (left 4 panels) clearly shows that our method achieves considerably better test errors across all nets and

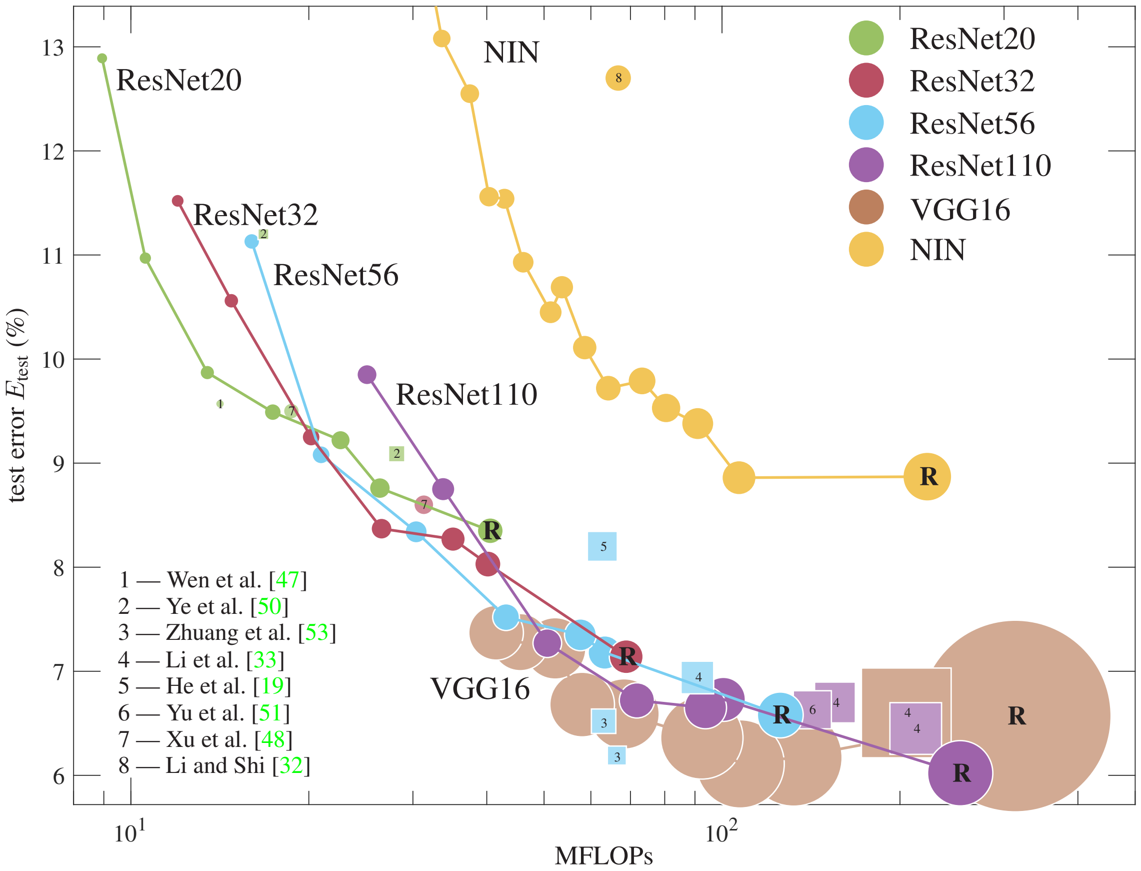

Figure 2. Error-compression space of test error (Y axis), inference FLOPs (X axis) and number of parameters (ball size for each net), for ResNets, VGG16 and NIN trained on CIFAR10. Results of our algorithm over different λ values for a given network span a curve, shown as connected circles ●—●, which starts on the lower right at the reference **R** ($\lambda = 0$) and then moves left and up. Other published results using low-rank compression are shown as isolated circles labeled with a citation. Other published results involving structured filter pruning for faster inference are shown as isolated squares labeled with a citation. Each color corresponds to a different reference net. The area of a circle or square is proportional to the number of parameters in the corresponding compressed model. Ideal models are small balls (having few parameters) on the left-bottom (where both error and FLOPs are the smallest).

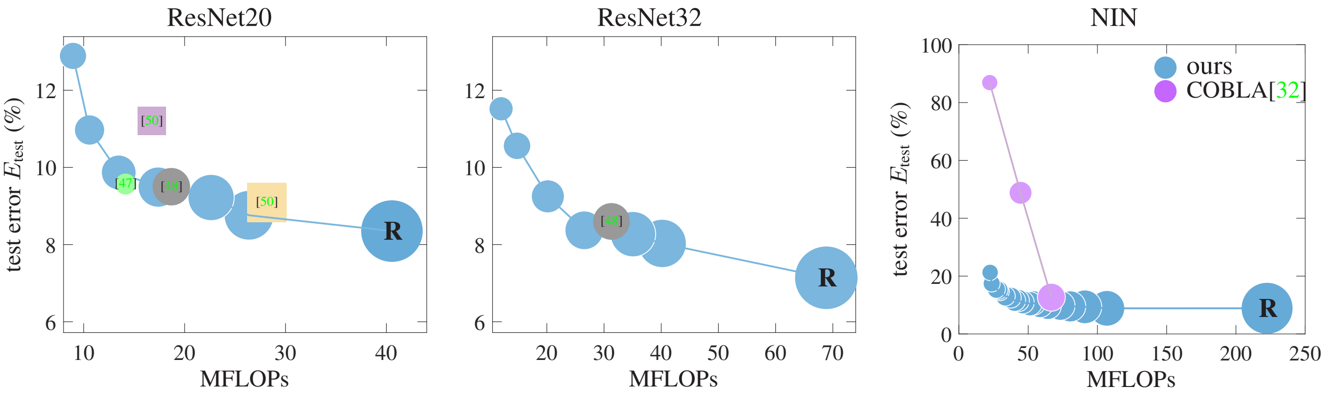

Figure 3. Enlarged portions of fig. 2 for ResNet20, 32, NIN. Blue circles correspond to our LC algorithm over various λ values, with area proportional to the number of parameters (independently normalized for each figure). Results of other compressions are given by other colored circles (low-rank compressions, in particular COBLA [32] in the NIN plot) and squares (filter pruning).

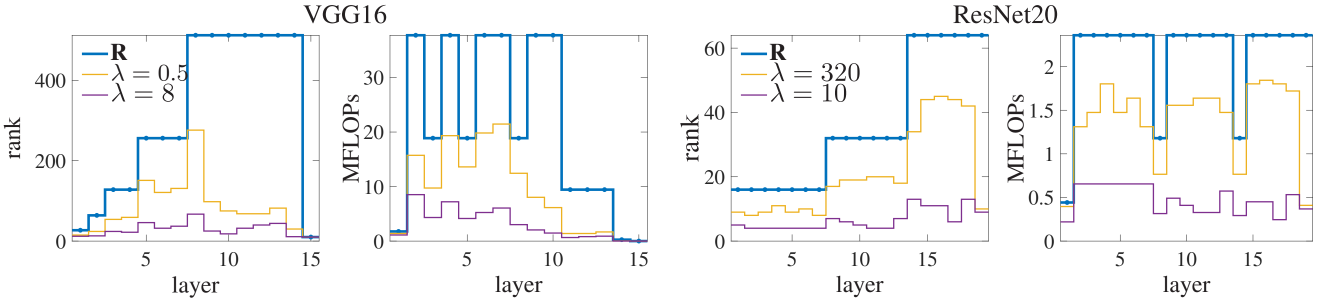

Figure 4. Distribution of ranks and FLOPs over layers for VGG16 and ResNet20 using our method for selected values of λ ($\times 10^{-4}$).

all target compression ratios of FLOPs. The improvement is very significant compared to the value of the reference net, particularly for larger FLOP ratios (more compression), and for the nets that are harder to compress (ResNets). For VGG we even beat the reference test error over a wide range. The improvement over the baselines happens because the latter irrevocably commit to the heuristically selected ranks, while our algorithm explores different rank selections eventually converging to a much better one. Fig. 1 (right) shows the selected ranks per layer for the case of a NIN (with ranks selected to achieve 2× speedup). The selection of our algorithm is quite different from the baselines and shows a complex pattern over the NIN layers. We discuss this further later. Fig. 1 (left panel for NIN) shows the results for the COBLA joint weights-and-ranks optimization [32]. It does much worse than our algorithm and even than the baselines, particularly in the high compression regime; see also fig. 2 and fig. 3 (right).

Comparing the performance of compressed neural networks is a task complex, as there are multiple dimensions for it that are not directly correlated. For example, the number of parameters is in general not proportional to the runtime (FLOPs) at inference. (Also, reported compression ratios of any kind can be easily inflated by compressing a large, overparameterized model in the first place.) Rather than reporting a single compression ratio as in fig. 1, in fig. 2 we show test error (Y axis), inference FLOPs (X axis) and number of parameters (ball size for each net), in order to understand the interplay between these for our algorithm and other published works. Thus, fig. 2 shows low-rank compression for all our results and others' published results [32, 47, 48]. Importantly, it also puts low-rank compression in perspective with other reported results for faster inference involving structured filter pruning [19, 33, 50, 51, 53]. Some portions of the figure (for ResNet20, 32 and NIN) are enlarged in fig. 3 to avoid clutter.

We see that, for a given network over various levels of low-rank compression, our algorithm's results are the best (with very few exceptions) in the Pareto curve sense: they cannot be improved in both error and FLOPs. If looking at specific nets, the optimal ones are ResNets and VGG, although which one is better depends on the area of the error-compression space we consider. For example, the reference VGG16 has many parameters (large ball) and is slow (large FLOPs) compared to reference ResNets (as is widely known). But scanning our low-rank curve over λ shows a different picture: low-rank VGG16s improve significantly over low-rank ResNets in some regions of the error-FLOPs space (e.g. a low-rank VGG16 achieves 6.11% error with 107 MFLOPs, while the reference ResNet110 has 6.02%

	$\lambda \times 10^{-4}$	# params.	MFLOPs	top-1	top-5
	R	62.3M	1139	42.49	19.54
scheme 1	0.05	40.7M	436	41.56	19.15
	0.15	18.2M	263	42.63	19.95
	0.17	14.2M	240	42.83	19.93
scheme 2	0.05	40.5M	324	**41.46**	**19.14**
	0.10	25.2M	236	41.81	19.40
	0.15	18.1M	190	42.07	19.54
	0.20	**12.4M**	**151**	42.69	19.83

Table 2. Our algorithm on AlexNet using low-rank parametrization schemes 1 and 2 (for several λs). We report: number of parameters and MFLOPs, and top-1/top-5 errors on the validation set (%).

	MFLOPs	top-1	top-5	ρ_{FLOPs}
Caffe-AlexNet [24]	724	42.70	**19.80**	1.00
Kim et al. [26], Tucker	272	n/a	21.67	2.66
Tai et al. [45], scheme 2	185	n/a	20.34	3.90
Wen et al. [47], scheme 1	269	n/a	20.14	2.69
Kim et al. [25], scheme 2	272	43.40	20.10	2.66
Yu et al. [51], filter prun.	232	44.13	n/a	3.12
Li et al. [34], filter prun.	334	43.17	n/a	2.16
Ding et al. [12], filter prun.	492	43.83	20.47	1.47
ours, scheme 1, $\lambda = 0.17$	240	42.83	19.93	3.01
ours, scheme 2, $\lambda = 0.20$	**151**	**42.69**	19.83	**4.79**

Table 3. AlexNet compression with our algorithm vs published work using low-rank methods and structured pruning. We report top-1/top-5 validation error (%) and MFLOPs number and ratio.

error with 252 MFLOPs). The low-rank models obtained with our algorithm are comparable and often considerably better than other published low-rank compression and structured pruning results.

With these results, we can also understand the important question of how the optimally selected ranks change over the layers of a net, and over λ values. In particular, is it possible to infer approximately optimal ranks beforehand by some heuristic, fix them and train the resulting net with SGD? Already the ranks found by our algorithm for NIN in fig. 1 show a complex pattern over layers that is quite different from those of the heuristics. Fig. 4 shows the rank distribution over layers obtained by our method for ResNet20 and VGG16 (for selected λ values), and the corresponding FLOPs/layer. The distribution is far from uniform and not directly related to either the maximum rank or the FLOPs at a layer. Some layers (e.g. 5 and 9 for VGG16) have much higher ranks than others. Their relative proportion does not stay the same for different λ values. For example, see layers 10 and 11 of VGG16: for $\lambda = 0.5$ the rank of layer 10 is greater than that of layer 11, but for $\lambda = 0.8$ the relation is reversed. These relations cannot be captured by simple heuristics, and need to be inferred via joint optimization, because the classification loss and the cost function (FLOPs, storage, etc.) interact in nonlinear ways. Understanding the rank distribution may be a useful research direction, as it may give insights about the design and structure of deep nets. In summary, we conclude that learning both ranks and weights is well worth the effort—which is not large: our LC algorithm is not much slower than such heuristic methods. It is simple to implement and its runtime (dominated by the L steps) is just a few times that of training the reference, full-rank net.

4.4. ImageNet experiments

We train a batch normalized version of the AlexNet network [29] with 62M parameters on the ImageNet ILSVRC 2012 dataset [42] using the augmentation procedure of the original paper. Our reference model achieves a validation accuracy of 57.71% (top-1) and 80.45% (top-5). We compress the reference network using our rank-selection algorithm for both low-rank decomposition schemes using various λ values (details in the suppl. mat.). See tables 2–3.

Our compressed networks achieve much better error and speedup (ρ_{FLOPs}) compared to other low-rank and filter pruning methods. Our smallest scheme-1 network has lower FLOPs and error than the scheme-1 decomposed AlexNet of [47]. Our smallest scheme-2 network achieves 4.79× FLOPs reduction wrt Caffe-AlexNet while having the same error, which outperforms similar scheme-2 methods of [25, 45], and structured pruning methods of [12, 34, 51].

5. Conclusion

We have approached the problem of low-rank compression in a multilayer deep net as a model selection problem of finding the ranks. The mathematical vehicle that makes this possible is the introduction of a model selection criterion that serves both to specify an application-driven cost of real interest, and to do an architecture search with guarantees of continued progress. Our resulting algorithm monotonically decreases at each iteration an objective function over both architectures (rank selections at each layer) and numerical weight values, switching from one architecture to another on the fly, and produces a good approximate solution to the problem. In our experiments, this finds solutions that improve over other ways of approximately determining the ranks, and which provide insights about the structure of deep nets depending on the types of layers. It also makes low-rank compression much more attractive for deep nets compared to pruning than previously thought.

Acknowledgements

Work partially supported by several GPU donations by the NVIDIA Corporation.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016)*, pages 265–283, Savannah, GA, Oct. 6–8 2016. 1

[2] Jose M. Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 856–867. MIT Press, Cambridge, MA, 2017. 2

[3] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, Dec. 2009. 2

[4] Miguel Á. Carreira-Perpiñán. Model compression as constrained optimization, with application to neural nets. Part I: General framework. arXiv:1707.01209, July 5 2017. 3

[5] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev. Model compression as constrained optimization, with application to neural nets. Part II: Quantization. arXiv:1707.04319, July 13 2017. 3

[6] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev. “Learning-compression” algorithms for neural net pruning. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’18)*, pages 8532–8541, Salt Lake City, UT, June 18–22 2018. 3

[7] Miguel Á. Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In Samuel Kaski and Jukka Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 10–19, Reykjavik, Iceland, Apr. 22–25 2014. 3

[8] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization, part 1: Low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4–5):249–429, 2016. 2

[9] Andrzej Cichocki, Anh-Huy Phan, Qibin Zhao, Namgil Lee, Ivan Oseledets, Masashi Sugiyama, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization, part 2: Applications and future perspectives. *Foundations and Trends in Machine Learning*, 9(6):431–673, 2017. 2

[10] Misha Denil, Babak Shakibi, Laurent Dinh, Marc’Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2148–2156. MIT Press, Cambridge, MA, 2013. 2

[11] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 1269–1277. MIT Press, Cambridge, MA, 2014. 2

[12] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive CNN width optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 1607–1616, Long Beach, CA, June 9–15 2019. 8

[13] Maryam Fazel. *Matrix Rank Minimization with Applications*. PhD thesis, Stanford University, Mar. 2002. 2

[14] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: Compressing convolutional and FC layers alike. arXiv:1611.03214, Nov. 10 2016. 2

[15] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2012. 1, 4

[16] Zaid Harchaoui, Matthijs Douze, Mattis Paulin, Miroslav Dudik, and Jérôme Malick. Large-scale image classification with trace-norm regularization. In *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’12)*, pages 3386–3393, Providence, RI, June 16–21 2012. 2

[17] Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009. 3

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’16)*, pages 770–778, Las Vegas, NV, June 26 – July 1 2016. 4, 6

[19] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proc. 17th Int. Conf. Computer Vision (ICCV’17)*, pages 1398–1406, Venice, Italy, Dec. 11–18 2017. 6, 7

[20] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi;. Deep Roots: Improving CNN efficiency with hierarchical filter groups. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’17)*, pages 1231–1240, Honolulu, HI, July 21–26 2017. 2

[21] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training CNNs with low-rank filters for efficient image classification. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016. 2

[22] Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *J. Multivariate Analysis*, 5(2):248–264, June 1975. 1

[23] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In Michel Valstar, Andrew French, and Tony Pridmore, editors, *Proc. of the 25th British Machine Vision Conference (BMVC 2014)*, Nottingham, UK, Sept. 1–5 2014. 2, 5

[24] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama,

and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014. 8

[25] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 12569–12577, Long Beach, CA, June 16–20 2019. 2, 8

[26] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016. 2, 8

[27] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. 2

[28] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Dept. of Computer Science, University of Toronto, Apr. 8 2009. 4

[29] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1106–1114. MIT Press, Cambridge, MA, 2012. 4, 8

[30] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015. 2

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998. 4

[32] Chong Li and C. J. Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Proc. 15th European Conf. Computer Vision (ECCV'18)*, pages 746–761, Munich, Germany, Sept. 8–14 2018. 2, 5, 6, 7

[33] Hao Li, Asim Kadav, Igor Durdanovic, and Hans P. Graf. Pruning filters for efficient ConvNets. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017. 6, 7

[34] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 7046–7055, Long Beach, CA, June 16–20 2019. 8

[35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *Int. Conf. Learning Representations (ICLR 2014)*, 2014. 4

[36] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–376, 1983. 4

[37] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006. 3

[38] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 442–450. MIT Press, Cambridge, MA, 2015. 2

[39] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Workshop on The Future of Gradient-Based Machine Learning Software (Autodiff)*, 2017. 1

[40] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, Aug. 2010. 2

[41] Gregory C. Reinsel and Raja P. Velu. *Multivariate Reduced-Rank Regression. Theory and Applications*. Number 136 in Lecture Notes in Statistics. Springer-Verlag, 1998. 1

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *Int. J. Computer Vision*, 115(3):211–252, Dec. 2015. 4, 8

[43] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arısoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'13)*, pages 6655–6659, Vancouver, Canada, Mar. 26–30 2013. 2

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015. 4

[45] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016. 2, 5, 8

[46] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized low rank models. *Foundations and Trends in Machine Learning*, 9(1):1–118, 2016. 2

[47] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *Proc. 17th Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Dec. 11–18 2017. 2, 5, 6, 7, 8

[48] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trained rank pruning for efficient deep neural networks. arXiv:1812.02402, Dec. 8 2018. 2, 5, 6, 7

[49] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In F. Bimbot, C. Cerisara, C. Fougeron, G. Gravier, L. Lamel, F. Pellegrino, and P. Perrier, editors, *Proc. of Interspeech'13*, pages 2365–2369, Lyon, France, Aug. 25–29 2013. 2

[50] Jianbo Ye, Xin Lu, Zhe Lin, and James Wang. Rethinking the smaller-norm-less-informative assumption in channel

pruning of convolution layers. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, Apr. 30 – May 3 2018. 6, 7

[51] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: Pruning networks using neuron importance score propagation. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pages 9194–9203, Salt Lake City, UT, June 18–22 2018. 6, 7, 8

[52] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, Oct. 2016. 2, 5

[53] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 815–886. MIT Press, Cambridge, MA, 2018. 6, 7