SPEEDING-UP CONVOLUTIONAL NEURAL NETWORKS USING FINE-TUNED CP-DECOMPOSITION

Vadim Lebedev[1,2], Yaroslav Ganin[1], Maksim Rakhuba[1,3], Ivan Oseledets[1,4], and Victor Lempitsky[1]

[1]Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russia
[2]Yandex, Moscow, Russia
[3]Moscow Institute of Physics and Technology, Moscow Region, Russia
[4]Institute of Numerical Mathematics RAS, Moscow, Russia

ABSTRACT

We propose a simple two-step approach for speeding up convolution layers within large convolutional neural networks based on tensor decomposition and discriminative fine-tuning. Given a layer, we use non-linear least squares to compute a low-rank CP-decomposition of the 4D convolution kernel tensor into a sum of a small number of rank-one tensors. At the second step, this decomposition is used to replace the original convolutional layer with a sequence of four convolutional layers with small kernels. After such replacement, the entire network is fine-tuned on the training data using standard backpropagation process.

We evaluate this approach on two CNNs and show that it is competitive with previous approaches, leading to higher obtained CPU speedups at the cost of lower accuracy drops for the smaller of the two networks. Thus, for the 36-class character classification CNN, our approach obtains a 8.5x CPU speedup of the whole network with only minor accuracy drop (1% from 91% to 90%). For the standard ImageNet architecture (AlexNet), the approach speeds up the second convolution layer by a factor of 4x at the cost of 1% increase of the overall top-5 classification error.

1 INTRODUCTION

Over the course of two years, Convolutional neural networks (CNNs) (LeCun et al., 1989) have revolutionized computer vision and became ubiquituous through a range of computer vision applications. In many ways, this breakthrough has become possible through the acceptance of new computational tools, most notably GPUs (Krizhevsky et al., 2012), but also CPU clusters (Dean et al., 2012) and FPGAs (Farabet et al., 2011). On the other hand, there is a rising interest to deploying CNNs on low-end architectures, such as standalone desktop/laptop CPUs, mobile processors, and CPU in robotics. On such processors, the computational cost of applying, yet alone training a CNN might pose a problem, especially when real-time operation is required.

The key layer of CNNs that distinguishes them from other neural networks and enables their recent success is the convolution operation. Convolutions dominate the computation cost associated with training or testing a CNN (especially for newer architectures such as (Simonyan & Zisserman, 2014), which tend to add more and more convolutional layers often at the expense of fully-connected layers). Consequently, there is a strong interest to the task of improving the efficiency of this operation (Chintala, 2014; Mathieu et al., 2013; Chellapilla et al., 2006). An efficient implementation of the convolution operation is one of the most important elements of all major CNN packages.

A group of recent works have achieved significant speed-ups of CNN convolutions by applying tensor decompositions. In more detail, recall that a typical convolution in a CNN takes as an input a 3D tensor (array) with the dimensions corresponding to the two spatial dimensions, and to different image maps. The output of a convolution is another similarly-structured 3D tensor. The convolution kernel itself constitutes a 4D tensor with dimensions corresponding to the two spatial dimensions, the input image maps, and the output image maps. Exploiting this tensor structure, previous works

(Denton et al., 2014; Jaderberg et al., 2014a) have suggested different tensor decomposition schemes to speed-up convolutions within CNNs. These schemes are applied to the kernel tensor and generalize previous 2D filter approximations in computer vision like (Rigamonti et al., 2013).

In this work, we further investigate tensor decomposition in the context of speeding up convolutions within CNNs. We use a standard (in tensor algebra) low-rank CP-decomposition (also known as PARAFAC or CANDECOMP) and an efficient optimization approach (non-linear least squares) to decompose the full kernel tensor.

We demonstrate that the use of the CP-decomposition on a full kernel tensor has the following important advantages:

- **Ease of the decomposition implementation.** CP-decomposition is one of the standard tools in tensor linear algebra with well understood properties and mature implementations. Consequently, one can use existing efficient algorithms such as NLS to compute it efficiently.
- **Ease of the CNN implementation.** CP-decomposition approximates the convolution with a 4D kernel tensor by the sequence of four convolutions with small 2D kernel tensors. All these lower dimensional convolutions represent standard CNN layers, and are therefore easy to insert into a CNN using existing CNN packages (no custom layer implementation is needed).
- **Ease of fine-tuning.** Related to the previous item, once a convolutional layer is approximated and replaced with a sequence of four convolutional layers with smaller kernels, it is straight-forward to fine-tune the *entire* network on training data using back-propagation.
- **Efficiency.** Perhaps most importantly, we found that the simple combination of the full kernel CP-decomposition and global fine-tuning can result in better speed-accuracy trade-off for the approximated networks than the previous methods.

The CNNs obtained in our method are somewhat surprisingly accurate, given that the number of parameters is reduced several times compared to the initial networks. Practically, this reduction in the number of parameters means more compact networks with reduced memory footprint, which can be important for architectures with limited RAM or storage memory. Such memory savings can be especially valuable for feed-forward networks that include convolutions with spatially-varying kernels ("locally-connected" layers).

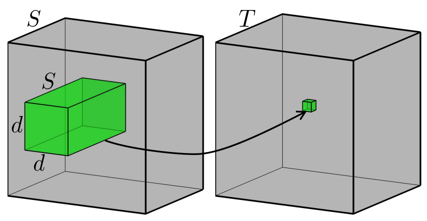
On the theoretical side, these results confirm the intuition that modern CNNs are over-parameterized, i.e. that the sheer number of parameters in the modern CNNs are not needed to store the information about the classification task but, rather, serve to facilitate convergence to good local minima of the loss function.

Below, we first discuss previous works of (Rigamonti et al., 2013; Jaderberg et al., 2014a; Denton et al., 2014) and outline the differences between them and our approach in Section 2. We then review the CP-decomposition and give the details of our approach in Section 3. The experiments on two CNNs, namely the character classification CNN from (Jaderberg et al., 2014a) and (Jaderberg et al., 2014b) and AlexNet from the `Caffe` package (Jia et al., 2014), follow in Section 4. We conclude with the summary and the discussion in Section 5.

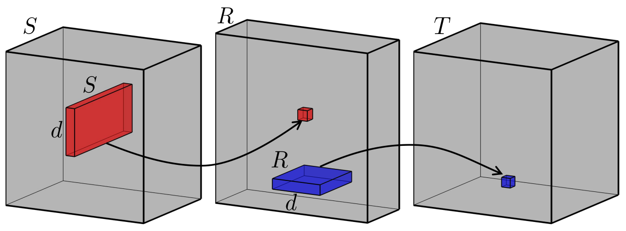
2 RELATED WORK

Decompositions for convolutions. Using low-rank decomposition to accelerate convolution was suggested by Rigamonti et al. (2013) in the context of codebook learning. In (Rigamonti et al., 2013), a bank of 2D or 3D filters X is decomposed into linear combinations of a shared bank of separable (decomposable) filters Y. The decompositions of filters within Y are independent (components are not shared).

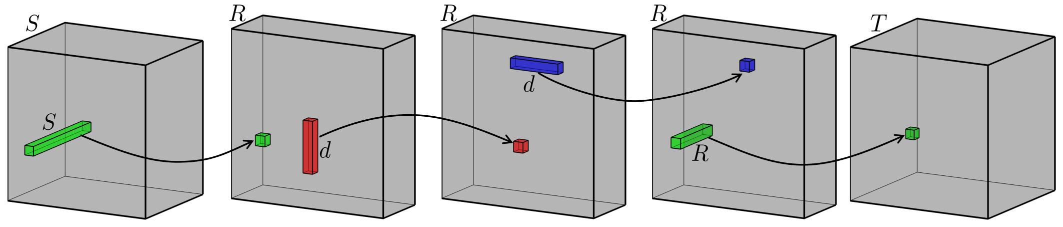
Jaderberg et al. (2014a) evaluated the decomposition (Rigamonti et al., 2013) in the context of CNNs and furthermore suggested a more efficient decomposition (Figure 1b) that effectively approximates the 4D kernel tensor as a composition (product) of two 3D tensors. In the experiments, Jaderberg et al. (2014a) have demonstrated the advantage of this scheme over (Rigamonti et al., 2013). In a sequel, when refering to (Jaderberg et al., 2014a) we imply this two-component decomposition.



(a) Full convolution



(b) Two-component decomposition (Jaderberg et al., 2014a)



(c) CP-decomposition

Figure 1: **Tensor decompositions for speeding up a generalized convolution.** Gray boxes correspond to 3D tensors (map stacks) within a CNN, with the two frontal sides corresponding to spatial dimensions. Arrows show linear mappings and demonstrate how scalar values on the right (small boxes corresponding to single elements of the target tensor) are computed. Initial **full convolution** (a) computes each element of the target tensor as a linear combination of the elements of a 3D subtensor that spans a spatial $d \times d$ window over all input maps. **Jaderberg et al. (2014a)** (b) approximate the initial convolution as a composition of two linear mappings with the intermediate map stack having R maps (where R is the rank of the decomposition). Each of the two mappings computes each target value based on a spatial window of size $1 \times d$ or $d \times 1$ in all input maps. Finally, **CP-decomposition** (c) used in our approach approximates the convolution as a composition of four convolutions with small kernels, so that a target value is computed based on a 1D-array that spans either one pixel in all input maps, or a 1D spatial window in one input map.

Once the decomposition is computed, Jaderberg et al. (2014a) perform "local" fine-tuning that minimizes the deviation between the full and the approximated convolutions outputs on the training data. Differently from Jaderberg et al. (2014a), our method fine-tunes the entire network based on the original discriminative criterion. While (Jaderberg et al., 2014a) reported that such discriminative fine-tuning was inefficient for their scheme, we found that in our case it works well, even when CP-decomposition has large approximation error. Below, we provide a theoretical complexity comparison and empirical comparison of our scheme with (Jaderberg et al., 2014a).

In the work that is most related to ours, Denton et al. (2014) have suggested a scheme based on the CP-decomposition of parts of the kernel tensor obtained by *biclustering* (alongside with a different decompositions for the first convolutional layer and the fully-connected layers). Biclustering of (Denton et al., 2014) splits the two non-spatial dimensions into subgroups, and reduces the effective ranks in the CP-decomposition. CP-decompositions of the kernel tensor parts in (Denton et al., 2014) have been computed with the greedy approach[1]. Our approach essentially simplifies that of Denton et al. (2014) in that we do not perform biclustering and apply CP-decomposition directly to the full convolution kernel tensor. On the other hand, we replace greedy computation of CP-decomposition with non-linear least squares. Finally, as discussed above, we fine-tune the complete network by backpropagation, whereas Denton et al. (2014) only fine-tunes the layers above the approximated one.

[1]Note that the alternating least squares process mentioned in (Denton et al., 2014) refers to computing the best next rank-1 tensor, but the outer process of adding rank-1 tensors is still greedy.

3 METHOD

Overall our method is a conceptually simple two-step approach: (1) take a convolutional layer and decompose its kernel using CP-decomposition, (2) fine-tune the entire network using backpropagation. If necessary, proceed to another layer. Below, we review the CP-decomposition, which is at the core of our method, and provide the details for the two steps of our approach.

3.1 CP-DECOMPOSITION REVIEW

Tensor decompositions are a natural way to generalize low-rank approach to multidimensional case. Recall that a low-rank decomposition of a matrix A of size $n \times m$ with rank R is given by:

$$A(i,j) = \sum_{r=1}^{R} A_1(i,r) A_2(j,r), \quad i = \overline{1,n}, \quad j = \overline{1,m}, \tag{1}$$

and leads to the idea of separation of variables. The most straightforward way to separate variables in case of many dimensions is to use the canonical polyadic decomposition (CP-decomposition, also called as CANDECOMP/PARAFAC model) (Kolda & Bader, 2009). For a d-dimensional array A of size $n_1 \times \dots \times n_d$ a CP-decomposition has the following form

$$A(i_1, \dots, i_d) = \sum_{r=1}^{R} A_1(i_1, r) \dots A_d(i_d, r), \tag{2}$$

where the minimal possible R is called canonical rank. The profit of this decomposition is that we need to store only $(n_1 + \dots + n_d)R$ elements instead of the whole tensor with $n_1 ... n_d$ elements.

In two dimensions, the low-rank approximation can be computed in a stable way by using singular value decomposition (SVD) or, if the matrix is large, by rank-revealing algorithms. Unfortunately, this is not the case for the CP-decomposition when $d > 2$, as there is no finite algorithm for determining canonical rank of a tensor (Kolda & Bader, 2009). Therefore, most algorithms approximate a tensor with different values of R until the approximation error becomes small enough. This leads to the point that for finding good low-rank CP-decomposition certain tricks have to be used. A detailed survey of methods for computing low-rank CP-decompositions can be found in (Tomasi & Bro, 2006). As a software package to calculate the CP-decomposition we used `Tensorlab` (Sorber et al., 2014). For our purposes, we chose non-linear least squares (NLS) method, which minimizes the L2-norm of the approximation residual (for a user-defined fixed R) using Gauss-Newton optimization.

Such NLS optimization is capable of obtaining much better approximations than the strategy of greedily finding best rank-1 approximation of the residual vectors used in Denton et al. (2014). The fact that the greedy rank-1 algorithm may increase tensor rank can be found in (Stegeman & Comon, 2010; Kofidis & Regalia, 2002). We also give a simple example highlighting this advantage of the NLS in the Appendix.

3.2 KERNEL TENSOR APPROXIMATION

CNNs (LeCun et al., 1989) are feed-forward multi-layer architectures that map the input images to certain output vectors using a sequence of operations. The units within CNN are organized as a sequence of 3D tensors (“map stacks”) with two spatial dimensions and the third dimension corresponding to different “maps” or ”channels”[2].

The most “important” and time-consuming operation within modern CNNs is the generalized convolution that maps an input tensor $U(\cdot,\cdot,\cdot)$ of size $X \times Y \times S$ into an output tensor $V(\cdot,\cdot,\cdot)$ of size $(X-d+1) \times (Y-d+1) \times T$ using the following linear mapping:

$$V(x,y,t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^{S} K(i-x+\delta, j-y+\delta, s, t)\, U(i,j,s) \tag{3}$$

[2]These tensors are 4D if/when a CNN is applied to a batch of images, with the fourth dimension corresponding to different images in a batch. This extra dimension does not affect the derivation below and is therefore disregarded.

Here, $K(\cdot,\cdot,\cdot,\cdot)$ is a 4D kernel tensor of size $d{\times}d{\times}S{\times}T$ with the first two dimensions corresponding to the spatial dimensions, the third dimension corresponding to different input channels, the fourth dimension corresponding to different output channels. The spatial width and height of the kernel are denoted as d, while δ denotes "half-width" $(d-1)/2$ (for simplicity we assume square shaped kernels and even d).

The rank-R CP-decomposition (2) of the 4D kernel tensor has the form:

$$K(i,j,s,t) = \sum_{r=1}^{R} K^x(i-x+\delta,r)\, K^y(j-y+\delta,r)\, K^s(s,r)\, K^t(t,r)\,, \tag{4}$$

where $K^x(\cdot,\cdot)$, $K^y(\cdot,\cdot)$, $K^s(\cdot,\cdot)$, $K^t(\cdot,\cdot)$ are the four components of the composition representing 2D tensors (matrices) of sizes $d{\times}R$, $d{\times}R$, $S{\times}R$, and $T{\times}R$ respectively.

Substituting (4) into (3) and performing permutation and grouping of summands gives the following expression for the approximate evaluation of the convolution (3):

$$V(x,y,t) = \sum_{r=1}^{R} K^t(t,r) \left(\sum_{i=x-\delta}^{x+\delta} K^x(i-x+\delta,r) \left(\sum_{j=y-\delta}^{y+\delta} K^y(j-y+\delta,r) \left(\sum_{s=1}^{S} K^s(s,r)\, U(i,j,s) \right)\right)\right) \tag{5}$$

Based on (5), the output tensor $V(\cdot,\cdot,\cdot)$ can be computed from the input tensor $U(\cdot,\cdot,\cdot)$ via a sequence of four convolutions with smaller kernels (Figure 1c):

$$U^s(i,j,r) = \sum_{s=1}^{S} K^s(s,r)\, U(i,j,s) \tag{6}$$

$$U^{sy}(i,y,r) = \sum_{j=y-\delta}^{y+\delta} K^y(j-y+\delta,r)\, U^s(i,j,r) \tag{7}$$

$$U^{syx}(x,y,r) = \sum_{i=x-\delta}^{x+\delta} K^x(i-x+\delta,r)\, U^{sy}(i,y,r) \tag{8}$$

$$V(x,y,t) = \sum_{r=1}^{R} K^t(t,r)\, U^{syx}(x,y,r)\,, \tag{9}$$

where $U^s(\cdot,\cdot,\cdot)$, $U^{sy}(\cdot,\cdot,\cdot)$, and $U^{syx}(\cdot,\cdot,\cdot)$ are intermediate tensors (map stacks).

3.3 Implementation and Fine-tuning

Computing $U^s(\cdot,\cdot,\cdot)$ from $U(\cdot,\cdot,\cdot)$ in (6) as well as $V(\cdot,\cdot,\cdot)$ from $U^{syx}(\cdot,\cdot,\cdot)$ in (9) represent so-called $1{\times}1$ convolutions (also used within "network-in-network" approach (Lin et al., 2013)) that essentially perform pixel-wise linear re-combination of input maps. Computing $U^{sy}(\cdot,\cdot,\cdot)$ from $U^s(\cdot,\cdot,\cdot)$ and $U^{syx}(\cdot,\cdot,\cdot)$ from $U^{sy}(\cdot,\cdot,\cdot)$ in (7) and (8) are "standard" convolutions with small kernels that are "flat" in one of the two spatial dimensions.

We use the popular `Caffe` package (Jia et al., 2014) to implement the resulting architecture, utilizing standard convolution layers for (7) and (8), and an optimized $1{\times}1$ convolution layers for (6) and (9). The resulting architecture is fine-tuned through standard backpropagation (with momentum) on training data. All network layers including layers above the approximated layer, layers below the approximated layer, and the four inserted convolutional layers participate in the fine-tuning. We found, however, that the gradients within the inserted layers are prone to gradient explosion, so one should either be careful to keep the learning rate low, or fix some or all of the inserted layers, while still fine-tuning layers above and below.

3.4 Complexity analysis

Initial convolution operation is defined by STd^2 parameters (number of elements in the kernel tensor) and requires the same number of "multiplication+addition" operations per pixel.

For (Jaderberg et al., 2014a) this number changes to $Rd(S+T)$, where R is the rank of the decomposition (see Figure 1b and Jaderberg et al. (2014a)). While the two numbers are not directly comparable, assuming that the required rank is comparable or several times smaller than S and T (e.g. taking $R \approx \frac{ST}{S+T}$), the scheme (Jaderberg et al., 2014a) gives a reduction in the order of d times compared to the initial convolution.

For (Denton et al., 2014) in the absence of bi-clustering as well as in the case of our approach, the complexity is $R(S+2d+T)$ (again, both for the number of parameters and for the number of "multiplications+additions" per output pixel). Almost always, $d \ll T$, which for the same rank gives a further factor of d improvement in complexity over (Jaderberg et al., 2014a) (and an order of d^2 improvement over the initial convolution when $R \approx \frac{ST}{S+T}$).

The bi-clustering in (Denton et al., 2014) makes a "theoretical" comparison with the complexity of our approach problematic, as on the one hand bi-clustering increases the number of tensors to be approximated, but on the other hand, reduces the required ranks considerably (so that assuming the same R would not be reasonable). We therefore restrict ourselves to the empirical comparison.

4 EXPERIMENTS

In this section we test our approach on two network architectures, small character-classification CNN and a bigger net trained for ILSVRC. Most of our experiments are devoted to the approximation of single layers, when other layers remain intact apart from the fine-tuning.

We make several measurements to evaluate our models. After the approximation of the kernel tensor with the CP-decomposition, we calculate the accuracy of this decomposition, i.e. $\|K' - K\|/\|K\|$, where K is the original tensor and K' is the obtained approximation. The difference between the original tensor and our approximation may disturb data propagation in CNN, resulting in the drop of classification accuracy. We measure this drop before and after the fine-tuning of CNN.

Furthermore, we record the CPU timings for our models and report the speed-up compared to the CPU timings of the original model (all timings are based on `Caffe` code run in the CPU mode on image batches of size 64). Finally, we report the reduction in the number of parameters resulting from the low-rank approximation. All results are reported for a number of ranks R.

4.1 CHARACTER-CLASSIFICATION CNN

We use use CNN described in (Jaderberg et al., 2014b) for our experiments. The network has four convolutional layers with maxout nonlinearities between them and a softmax output. It was trained to classify 24×24 image patches into one of 36 classes (10 digits plus 26 characters). Our `Caffe` port of the publicly available pre-trained model (refered below as `CharNet`) achieves 91.2% accuracy on test set (very similar to the original).

As in (Jaderberg et al., 2014a), we consider only the second and the third convolutional layers, which constitute more then 90% of processing time. Layer 2 has 48 input and 128 output channels and filters of size 9×9, layer 3 has 64 input and 512 output channels, filter size is 8×8.

The results of separate approximation of layers 2 and 3 are shown in figures 2a and 2b. Tensor approximation error diminishes with the growth of approximation rank, and when the approximation rank becomes big enough, it is possible to approximate weight tensor accurately. However, our experiments showed that accurate approximation is not required for the network to function properly. For example, while approximating layer 3, network classification accuracy is unaffected even if tensor approximation error is as big as 78%.

Combining approximations. We have applied our methods to two layers of the network using the following procedure. Firstly, layer 2 was approximated with rank 64. After that, the drop in accuracy was made small by fine-tuning of all layers but the new ones. Finally, layer 3 was approximated with rank 64, and for this layer such approximation does not result in significant drop of network prediction accuracy, so there is no need to fine-tune the network one more time.

The network derived by this procedure is 8.5 times faster than original model, while classification accuracy drops by 1% to 90.2%. Comparing with (Jaderberg et al., 2014a), we achieve almost two times bigger speedup for the same loss of accuracy, ((Jaderberg et al., 2014a) incurs 1% accuracy loss for the speedup of 4.2x and 5% accuracy loss for the speedup of 6x).

4.2 AlexNet

Following Denton et al. (2014) we also consider the second convolutional layer of `AlexNet` Krizhevsky et al. (2012). As a baseline we use a pre-trained model shipped with `Caffe`. We summarize various network properties for several different ranks of approximation in Figure 2c. It can be noticed that `conv2` of the considered network demands far larger rank (comparing to the `CharNet` experiment) for achieving proper performance.

Overall, in order to reach the 0.5% accuracy drop reported in (Jaderberg et al., 2014b) it is sufficient to take 200 components, which also gives a superior layer speed-up ($3.6\times$ vs. $2\times$ achieved by Scheme 2 of (Jaderberg et al., 2014b)). The running time of the `conv2` can be further reduced if we allow for slightly more misclassifications: rank 140 approximation leads to $4.5\times$ speed-up at the cost of $\approx 1\%$ accuracy loss surpassing the results of (Denton et al., 2014).

Along with conventional full-network fine-tuning we tried to refine the obtained tensor approximation by applying the data reconstruction approach from (Jaderberg et al., 2014b). Unfortunately, we failed to find a good SGD learning rate: larger values led to the exploding gradients, while the smaller ones did not allow to sensibly reduce the reconstruction loss. We suspect that this effect is due to the instability of the low-rank CP-decomposition (De Silva & Lim, 2008). One way to circumvent the issue would be to alternate the components learning (i.e. not optimizing all of them at once), which is the scope of our future work.

Our latest experiments showed that while our approach is superior for smaller architectures (as in character classification), it is not the best one for large nets such as AlexNet. Although (Jaderberg et al., 2014b) mentions that application of their approach to the second layer of the OverFeat architecture yields a $2\times$ speed-up, our colleagues at Yandex have discovered that a far greater speed-up with (Jaderberg et al., 2014b) can be reached, at least for AlexNet. In particular, our experiments with (Jaderberg et al., 2014b) on AlexNet have showed that the second convolutional layer of AlexNet can be accelerated by the factor of $6.6\times$ at the cost of $\approx 1\%$ accuracy loss via Scheme 2 and data optimization as described in (Jaderberg et al., 2014b) (as compared to $4.5\times$ for similar accuracy loss obtainable with our method).

4.3 NLS vs. Greedy

One of our main contributions is pointing out that greedy CP-decomposition works worse than more advanced algorithms such as non-linear least squares (NLS), and evaluate this degradation in the context of speeding up CNNs.

We perform comparisons on the second layers of both `CharNet` and `AlexNet`. For `CharNet`, we also evaluate the combination with fine-tuning. Furthermore, for `CharNet`, we also tried initializing the layers randomly (using the scheme of (Glorot & Bengio, 2010)).

The results in Table 1 clearly demonstrate two related things. Firstly, NLS decomposition leads to significantly higher accuracy whether with fine-tuning or without. The advantage is greater for the more complex network (`AlexNet`).

Secondly, the output of the fine-tuning clearly depends on the quality of the approximation. This observation concurs with the hypothesis that the large number of parameters in CNN is needed to avoid poor local minima. Indeed, CP-decomposition radically decreases the number of parameters in a layer. While good minima may still exist (e.g. the NLS+FT result), optimization is prone to stucking in a much worse minima (e.g. the Random+FT result).

5 Discussion

We have demonstrated that a rather straightforward application of a modern tensor decomposition method (NLS-based low-rank CP-decomposition) combined with the discriminative fine-tuning of the entire network can achieve considerable speedups with minimal loss in accuracy.

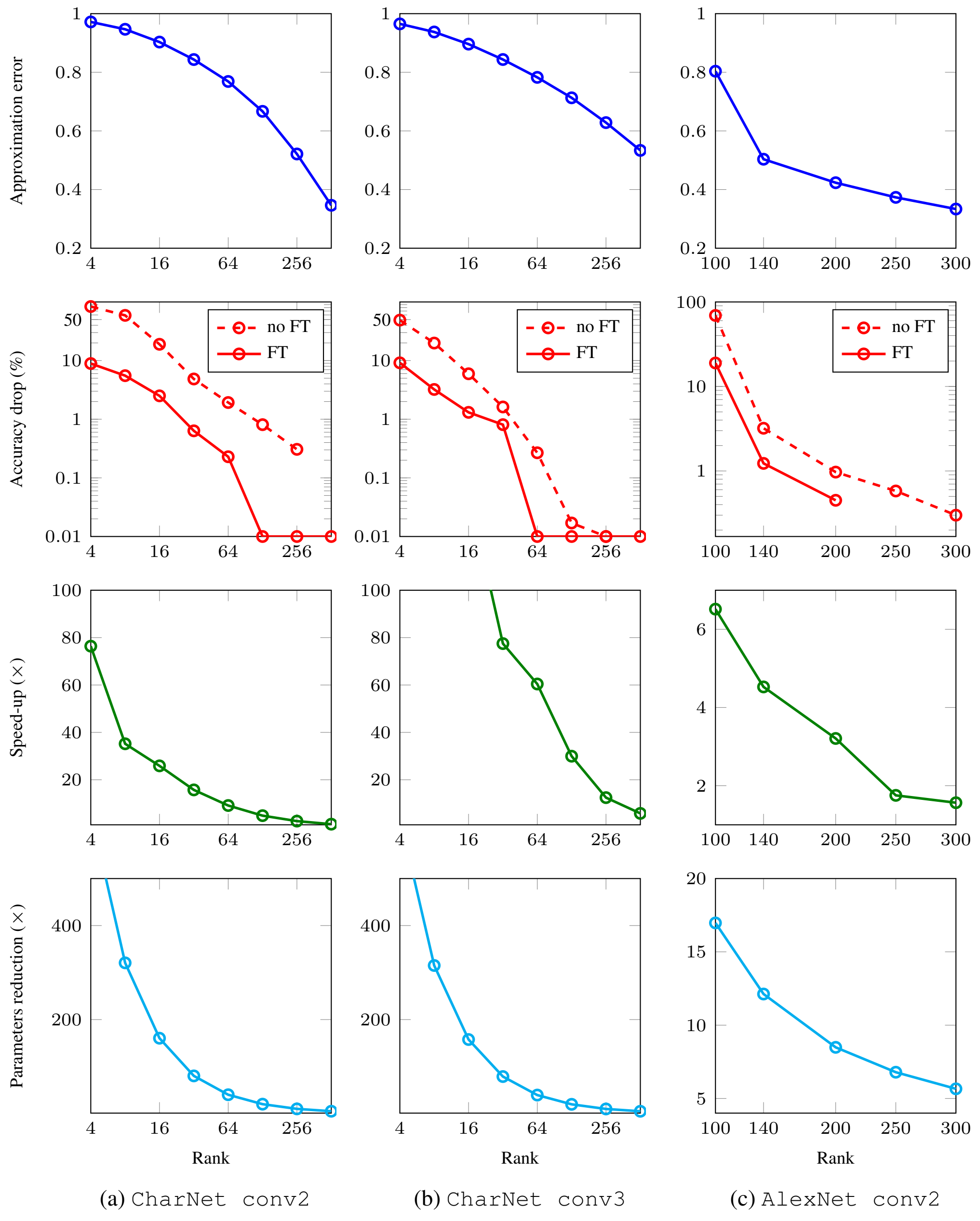


Figure 2: **Various properties and performance metrics of different approximated CNNs plotted as functions of the approximation rank. First row:** kernel tensor approximation error. **Second row:** drop of the classification accuracy of the full model with approximated layers w.r.t the accuracy of the original model; *dashed* lines correspond to the non-tuned CNNs, *solid* lines depict the performance after the fine-tuning. *Note the log-scale. Cases, where the accuracy has actually improved are plotted at the bottom line.* **Third row:** empirical speed-up of the approximated layer w.r.t. the original layer. **Fourth row:** ratio between the numbers of parameters in the original and the approximated layers.

Table 1: **Accuracy drop for the greedy and the non-linear least-squares (NLS) CP-decomposition.** The results are given for the second layers of `CharNet` and `AlexNet`, for different decomposition ranks R, and the numbers correspond to the accuracy drop of the entire CNN. Original networks achieve 91.24% (`CharNet`) and 79.95% (`AlexNet`). For `CharNet` we also evaluate the effect of fine-tuning (FT), and also for the random initialization. NLS computation of the CP-decomposition invariably leads to better performance, and the advantage persists through fine-tuning.

	`CharNet`, R=16		`CharNet`, R=64		`CharNet`, R=256		`AlexNet`	`AlexNet`	`AlexNet`
	NO FT	FT	NO FT	FT	NO FT	FT	R=140	R=200	R=300
RANDOM	–	9.70	–	7.64	–	6.13	–	–	–
GREEDY	24.15	2.64	4.99	0.46	1.14	-0.31	65.04	7.29	4.76
NLS	**18.96**	**2.16**	**1.93**	**0.09**	**0.31**	**-0.52**	**3.21**	**0.97**	**0.30**

In the preliminary comparisons, this approach outperforms the previous methods of (Denton et al., 2014) and (Jaderberg et al., 2014a), for the character classification CNN. However, more comparisons especially with a more related work of (Denton et al., 2014) are needed. In particular, it is to be determined whether bi-clustering is useful, when non-linear least squares are used for CP-decomposition.

Another avenue of research are layers with spatially-varying kernels, such as used e.g. in (Taigman et al., 2014). Firstly, these layers would greatly benefit from the reduction in the number of parameters. Secondly, the spatial variation of the kernel might be embedded into extra tensor dimensions, which may open up further speed-up possibilities.

Finally, similarly to (Denton et al., 2014), we note that low-rank decompositions seems to have a regularizing effects allowing to slightly improve the overall accuracy for higher rank.

ACKNOWLEDGMENTS

The work presented in Section 3 was supported by Russian Science Foundation grant 14-11-00659.

REFERENCES

Chellapilla, Kumar, Puri, Sidd, and Simard, Patrice. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

Chintala, Soumith. Convnet-benchmarks. `https://github.com/soumith/convnet-benchmarks`, 2014. Accessed: 2014-12-19.

De Silva, Vin and Lim, Lek-Heng. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.*, 30(3):1084–1127, 2008.

Dean, Jeffrey, Corrado, Greg, Monga, Rajat, Chen, Kai, Devin, Matthieu, Mao, Mark, Senior, Andrew, Tucker, Paul, Yang, Ke, Le, Quoc V, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

Denton, Emily, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. *arXiv preprint arXiv:1404.0736*, 2014.

Farabet, Clément, LeCun, Yann, Kavukcuoglu, Koray, Culurciello, Eugenio, Martini, Berin, Akselrod, Polina, and Talay, Selcuk. Large-scale FPGA-based convolutional networks. *Machine Learning on Very Large Data Sets*, 2011.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and*

Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010, pp. 249–256, 2010.

Jaderberg, Max, Vedaldi, Andrea, and Zisserman, Andrew. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014a.

Jaderberg, Max, Vedaldi, Andrea, and Zisserman, Andrew. Deep features for text spotting. In *Computer Vision–ECCV 2014*, pp. 512–528. Springer, 2014b.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Kofidis, Eleftherios and Regalia, Phillip A. On the best rank-1 approximation of higher-order supersymmetric tensors. *SIAM J. Matrix Anal. Appl.*, 23(3):863–884, 2002.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann, Boser, Bernhard, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne, and Jackel, Lawrence D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Mathieu, Michael, Henaff, Mikael, and LeCun, Yann. Fast training of convolutional networks through FFTs. *arXiv preprint arXiv:1312.5851*, 2013.

Rigamonti, Roberto, Sironi, Amos, Lepetit, Vincent, and Fua, Pascal. Learning separable filters. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 2754–2761. Ieee, 2013.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Sorber, L., Van Barel, M., and De Lathauwer, L. Tensorlab v2.0. *Available online*, 2014. URL `http://tensorlab.net`.

Stegeman, Alwin and Comon, Pierre. Subtracting a best rank-1 approximation may increase tensor rank. *Linear Algebra Appl.*, 433(7):1276–1300, 2010.

Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1701–1708. IEEE, 2014.

Tomasi, Giorgio and Bro, Rasmus. A comparison of algorithms for fitting the parafac model. *Comp. Stat. Data An.*, 50(7):1700–1734, 2006.

Appendix

In all our experiments, we consistently found that non-linear least squares (NLS) with the implementation from (Sorber et al., 2014) yield better CP-decompositions with smaller ranks (for the same approximation accuracy) than the greedy approach of finding the next best rank-one tensor and adding it to the decomposition.

Such advantage can be highlighted by the following example of $2 \times 2 \times 2$ tensor G of rank two, whose frontal slices are defined by:

$$G_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad G_2 = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}.$$

We checked numerically that a sequence of two best rank-one approximations fails to approximate the tensor G – relative error was 0.35. In contrast to that, the best rank-two approximation (with 10^{-7} error in this case) was successfully obtained by the NLS optimization.