Ultimate tensorization: convolutions and FC alike

Timur Garipov[1] **Dmitry Podoprikhin**[1,2] **Alexander Novikov**[3,4] **Dmitry Vetrov**[2,3]
[1]Moscow State University, Moscow, Russia
[2]Yandex, Moscow, Russia
[3]National Research University Higher School of Economics, Moscow, Russia
[4]Institute of Numerical Mathematics of the Russian Academy of Sciences, Moscow, Russia
timgaripov@gmail.com podoprikhin.dmitry@gmail.com
novikov@bayesgroup.ru vetrovd@yandex.ru

Abstract

Convolutional neural networks excel in image recognition tasks, but this comes at the cost of high computational and memory complexity. To tackle this problem, [1] developed a tensor factorization framework to compress fully-connected layers. In this paper, we focus on compressing convolutional layers. We show that while the direct application of the tensor framework [1] to the 4-dimensional kernel of convolution does compress the layer, we can do better. We reshape the convolutional kernel into a tensor of higher order and factorize it. We combine the proposed approach with the previous work to compress both convolutional and fully-connected layers of a network and achieve $80\times$ network compression rate with 1.1% accuracy drop on the CIFAR-10 dataset.

1 Introduction

Convolutional Neural Networks (CNNs) show state-of-the-art performance on many problems in computer vision, natural language processing and other fields [2, 3]. At the same time, CNNs require millions of floating point operations to process an image and therefore real-time applications need powerful CPU or GPU devices. Moreover, these networks contain millions of trainable parameters and consume hundreds of megabytes of storage and memory bandwidth [4]. Thus, CNNs are forced to use RAM instead of solely relying on the processor cache – orders of magnitude more energy efficient memory device [5] – which increases the energy consumption even more. These reasons restrain the spread of CNNs on mobile devices.

To address the storage and memory requirements of neural networks, [1] used tensor decomposition techniques to compress fully-connected layers. They represented the parameters of the layers in the Tensor Train format [6] and learned the network from scratch in this representation. This approach provided enough compression to move the storage bottleneck of VGG-16 [5] from the fully-connected layers to convolutional layers. For a more detailed literature overview, see Sec. 5.

In this paper, we propose a tensor factorization based method to compress convolutional layers. Our contributions are:

- We experimentally show that applying the Tensor Train decomposition – the compression technique used in [1] – directly to the tensor of a convolution yields poor results (see Sec. 6). We explain this behavior and propose a way to reshape the 4-dimensional kernel of a convolution into a multidimensional tensor to fully utilize the compression power of the Tensor Train decomposition (see Sec. 4).
- We experimentally show that the proposed approach allows compressing a network that consists only of convolutions up to $4\times$ times with 2% accuracy decrease (Sec. 6).
- We combine the proposed approach with the fully-connected layers compression of [1]. Compressing both convolutional and fully-connected layers of a network yields $82\times$ network compression with 1% accuracy drop, see Sec. 6.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2 Convolutional Layer

A convolutional network is a type of feed-forward architecture that transforms an input image to the final class scores using a sequence of layers. The main building block of such networks is a convolutional layer, that transforms the 3-dimensional input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{W \times H \times C}$ into the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{W-l+1 \times H-l+1 \times S}$ by *convolving* $\boldsymbol{\mathcal{X}}$ with the kernel tensor $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{\ell \times \ell \times C \times S}$:

$$\mathcal{Y}(x, y, s) = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \sum_{c=1}^{C} \mathcal{K}(i, j, c, s)\, \mathcal{X}(x + i - 1, y + j - 1, c). \tag{1}$$

To improve the computational performance, many deep learning frameworks reduce the convolution (1) to a matrix-by-matrix multiplication [7, 8]. We exploit this matrix formulation to motivate a particular way of applying the Tensor Train format to the convolutional kernel (see Sec. 4). In the rest of this section, we introduce the notation needed to reformulate convolution (1) as a matrix-by-matrix multiplication $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{K}$.

For convenience, we denote $H' = H - \ell + 1$ and $W' = W - \ell + 1$. Let us reshape the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{W' \times H' \times S}$ into a matrix $\boldsymbol{Y}$ of size $W'H' \times S$ in the following way

$$\mathcal{Y}(x, y, s) = Y(x + W'(y - 1), s).$$

Let us introduce a matrix $\boldsymbol{X}$ of size $W'H' \times \ell^2 C$, the k-th row of which corresponds to the $\ell \times \ell \times C$ patch of the input tensor that is used to compute the k-th row of the matrix $\boldsymbol{Y}$

$$\mathcal{X}(x + i - 1, y + j - 1, c) = X(x + W'(y - 1), i + \ell(j - 1) + \ell^2(c - 1)),$$

where $y = 1, \ldots, H'$, $x = 1, \ldots, W'$, $i, j = 1, \ldots, \ell$. Finally, we reshape the kernel tensor $\boldsymbol{\mathcal{K}}$ into a matrix $\boldsymbol{K}$ of size $\ell^2 C \times S$

$$\mathcal{K}(i, j, c, s) = K(i + \ell(j - 1) + \ell^2(c - 1), s).$$

Using the matrices defined above, we can rewrite the convolution definition (1) as $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{K}$.

Note that the compression approach presented in the rest of the paper works with other types of convolutions, such as convolutions with padding, stride larger than 1, or rectangular filters. But for clarity, we illustrate the proposed idea on the basic convolution (1).

3 Tensor Train Decomposition

The TT-decomposition (or TT-representation) of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ is the set of matrices $\boldsymbol{G}_k[j_k] \in \mathbb{R}^{r_{k-1} \times r_k}$, where $j_k = 1, \ldots, n_k$, $k = 1, \ldots, d$, and $r_0 = r_d = 1$, such that each of the tensor elements can be represented as

$$\mathcal{A}(j_1, j_2, \ldots, j_d) = \boldsymbol{G}_1[j_1]\boldsymbol{G}_2[j_2] \ldots \boldsymbol{G}_d[j_d]. \tag{2}$$

The elements of the collection $\{r_k\}_{k=0}^{d}$ are called *TT-ranks*. The collections of matrices $\{\{\boldsymbol{G}_k[j_k]\}_{j_k=1}^{n_k}\}_{k=1}^{d}$ are called *TT-cores* [6].

The TT-format requires $\sum_{k=1}^{d} n_k r_{k-1} r_k$ parameters to represent a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ which has $\prod_{k=1}^{d} n_k$ elements. The TT-ranks r_k control the trade-off between the number of parameters versus the accuracy of the representation: the smaller the TT-ranks, the more memory efficient the TT-format is.

For a matrix – a 2-dimensional tensor – the TT-decomposition coincides with the matrix low-rank decomposition. To represent a matrix more compactly than in the low-rank format, the matrix TT-format is defined in a special way. Let us consider a matrix $\boldsymbol{A}$ of size $M \times N$, where $M = \prod_{k=1}^{d} m_k$, $N = \prod_{k=1}^{d} n_k$, and reshape it into a tensor $\boldsymbol{\mathcal{A}}$ of size $n_1 m_1 \times n_2 m_2 \times \ldots \times n_d m_d$ by defining bijective mappings $\boldsymbol{\mu}(\ell) = (\mu_1(\ell), \ldots, \mu_d(\ell))$ and $\boldsymbol{\nu}(t) = (\nu_1(t), \ldots, \nu_d(t))$. The mapping $\boldsymbol{\mu}(\cdot)$ maps row index $\ell = 1, \ldots, M$ into a d-dimensional vector index, where k-th dimension $\mu_k(\cdot)$ varies from 1 to m_k. The bijection $\boldsymbol{\nu}(\cdot)$ maps column index $t = 1, \ldots, N$ into a d-dimensional vector index, where k-th dimension $\nu_k(\cdot)$ varies from 1 to n_k. Thus, using these mappings, we can form the tensor $\boldsymbol{\mathcal{A}}$, whose k-th dimension is indexed by the compound index $(\mu_k(\cdot), \nu_k(\cdot))$, and consider its TT-representation:

$$A(\ell, t) = \mathcal{A}((\mu_1(\ell), \nu_1(t)), \ldots, (\mu_d(\ell), \nu_d(t))) = \boldsymbol{G}_1[(\mu_1(\ell), \nu_1(t))] \ldots \boldsymbol{G}_d[(\mu_d(\ell), \nu_d(t))].$$

4 TT-convolutional Layer

In this section, we propose two ways to represent a convolutional kernel $\mathcal{K}$ in the TT-format. One way is to apply the TT-decomposition to the tensor $\mathcal{K}$ directly. To see the drawbacks of this approach, consider a 1×1 convolution, which is a small fully-connected layer applied to the channels of the input image in each pixel location. The kernel of such convolution is essentially a 2-dimensional array, and the TT-decomposition of 2-dimensional arrays coincides with the matrix low-rank format. But for fully-connected layers, the matrix TT-format proved to be more efficient than the matrix low-rank format [1]. Thus, we seek for a decomposition that would coincide with the matrix TT-format on 1×1 convolutions.

Taking into account that a convolutional layer can be formulated as a matrix-by-matrix multiplication (see Sec. 2), we reshape the 4-dimensional kernel tensor into a matrix $\boldsymbol{K}$ of size $\ell^2 C \times S$, where $\mathcal{K}(x, y, c, s) = K(\ell(y-1) + x + \ell^2(c-1), s)$. Then we apply the matrix TT-format (see Sec. 3) to the matrix $\boldsymbol{K}$, i.e. reshape it into a tensor $\widetilde{\mathcal{K}}$ and convert it into the TT-format. To reshape the matrix $\boldsymbol{K}$ into a tensor, we assume that its dimensions factorize: $C = \prod_{i=1}^{d} C_i$ and $S = \prod_{i=1}^{d} S_i$. Then we can define a $(d+1)$-dimensional tensor, where k-th dimension has the length $C_k S_k$ for $k = 1, \ldots, d$ and ℓ^2 for $k = 0$. Thus we obtain the following representation of the matrix $\boldsymbol{K}$

$$K(x + \ell(y-1) + \ell^2(c'-1), s') = \widetilde{\mathcal{K}}(\,(x + \ell(y-1), 1),\, (c_1, s_1), \ldots,\, (c_d, s_d)) = \\ = \widetilde{\boldsymbol{G}}_0[x + \ell(y-1), 1]\boldsymbol{G}_1[c_1, s_1] \ldots \boldsymbol{G}_d[c_d, s_d], \quad (3)$$

where $c' = c_1 + \sum_{i=2}^{d}(c_i - 1)\prod_{j=1}^{i-1} C_j$ and $s' = s_1 + \sum_{i=2}^{d}(s_i - 1)\prod_{j=1}^{i-1} S_j$.

To simplify the notation, we index the 0-th core $\widetilde{\boldsymbol{G}}_0$ with x and y: $\boldsymbol{G}_0[x, y] = \widetilde{\boldsymbol{G}}_0[\ell(y-1) + x, 1]$, where $x, y = 1, \ldots, \ell$. Finally, substituting $\widetilde{\boldsymbol{G}}_0$ into (3), we obtain the following decomposition of the convolution kernel $\mathcal{K}$

$$\mathcal{K}(x, y, c', s') = \boldsymbol{G}_0[x, y]\boldsymbol{G}_1[c_1, s_1] \ldots \boldsymbol{G}_d[c_d, s_d]. \quad (4)$$

To summarize our pipeline starting from an input tensor $\boldsymbol{\mathcal{X}}$ (an image): the TT-convolutional layer firstly reshapes the input tensor into a $(2+d)$-dimensional tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ of size $W \times H \times C_1 \times \ldots \times C_d$; then, the layer transforms the input tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ into the output tensor $\widetilde{\boldsymbol{\mathcal{Y}}}$ of size $(W - \ell + 1) \times (H - \ell + 1) \times S_1 \ldots \times S_d$ in the following way

$$\widetilde{\mathcal{Y}}(x, y, s_1, \ldots, s_d) = \sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\sum_{c_1, \ldots, c_d} \widetilde{\boldsymbol{\mathcal{X}}}(i + x - 1, j + y - 1, c_1, \ldots, c_d) \\ \boldsymbol{G}_0[i, j]\boldsymbol{G}_1[c_1, s_1] \ldots \boldsymbol{G}_d[c_d, s_d].$$

Note that for a 1×1 convolution ($\ell = 1$), the x and y indices vanish from the decomposition (4). The convolutional kernel collapses into a matrix $\{\mathcal{K}(1, 1, c', s')\}_{c'=1, s'=1}^{C, S}$ and the decomposition (4) for this matrix coincides with the Tensor Train format for the fully-connected layer proposed in [1].

To train a network with TT-conv layers, we treat the elements of the TT-cores as the parameters of the layer and apply stochastic gradient descent with momentum to them. To compute the necessary gradients we use automatic differentiation implemented in TensorFlow [9].

5 Related Work

Fully-connected layers of neural networks are traditionally considered as the memory bottleneck and numerous works focused on compressing these layers [10, 11, 1, 12]. However, several state-of-the-art neural networks are either bottlenecked by convolutional layers [13, 14], or their fully-connected layers can be compressed to move the bottleneck to the convolutional layers [1]. This leads to a number of works focusing on compressing and speeding up the convolutional layers [5, 15, 16, 17, 18, 19].

One approach to compressing a convolutional layer is based on either pruning less important weights from the convolutional kernel, or restricting possible variation of the weights (quantization), or both [5, 15, 17]. Our approach is compatible with the quantization technique: one can quantize the elements of the TT cores of the decomposition. Some works also add Huffman coding on top of other compression techniques [5], which is also compatible with the proposed method.

Another approach is to use tensor or matrix decompositions. CP-decomposition [18], Kronecker product factorization [19], and Tucker decomposition [20] allow to compress the network and/or speed up the inference time of convolutions.

Table 1: Compressing convolutional networks on the CIFAR-10 dataset (see Sec. 6). Different rows with the same model name correspond to different choices of the TT-ranks.

(a) Compressing the first baseline ('conv'), which is dominated by convolutions. 'TT-conv': the proposed compression method; 'TT-conv (naive)': direct application of the TT-decomposition to convolutional kernels.

Model	top-1 acc.	compr.
conv (baseline)	90.7	1
TT-conv	89.9	2.02
TT-conv	89.2	2.53
TT-conv	89.3	3.23
TT-conv	88.7	4.02
TT-conv (naive)	88.3	2.02
TT-conv (naive)	87.6	2.90

(b) Compressing the second baseline ('conv-fc'), which is dominated by fully-connected layers. 'conv-TT-fc': only the fully-connected part of the network is compressed; 'TT-conv-TT-fc': fully-connected and convolutional parts are compressed.

Model	top-1 acc.	compr.
conv-fc (baseline)	90.5	1
conv-TT-fc	90.3	10.72
conv-TT-fc	89.8	19.38
conv-TT-fc	89.8	21.01
TT-conv-TT-fc	90.1	9.69
TT-conv-TT-fc	89.7	41.65
TT-conv-TT-fc	89.4	82.87

6 Experiments

We evaluated [21] the compressing strength of the proposed approach on CIFAR-10 dataset [22]. We used two architectures as references: the first one is dominated by the convolutions (they occupy 99.54% parameters of the network), and the second one is dominated by the fully-connected layers (they occupy 95.98% parameters of the network).

Convolutional network. The first network has the following architecture: conv (64 output channels); BN; ReLU; conv (64 output channels); BN; ReLU; max-pool (3×3 with stride 2); conv (128 output channels); BN; ReLU; conv (128 output channels); BN; ReLU; max-pool (3×3 with stride 2); conv (128 output channels); BN; ReLU; conv (128 output channels); avg-pool (4×4); fc (128×10), where 'BN' stands for batch normalization [23] and all convolutional filters are of size 3×3. To compress the network we replace each convolutional layer excluding the first one (it contains less than 1% of the network parameters) with the TT-conv layer (see Sec. 4). For training, we initialize the TT-cores of the TT-conv layers with random noise and train the whole network from scratch.

We compare the proposed TT-convolution against the naive approach – directly applying the TT-decomposition to the 4-dimensional convolutional kernel (see Sec.4). We report that on the $2\times$ compression level the proposed approach (0.8% loss of accuracy) outperforms the naive baseline (2.4% loss of accuracy), for details see Tbl. 1a.

Network with convolutions and fully-connected layers. The second reference network was obtained from the first one by replacing the average pooling with two fully-connected layers of size 8192×1536 and 1536×512.

To compress the second network, we replace all layers excluding the first and the last one (they occupy less than 1% of parameters) with TT-conv and TT-fc [1] layers. To speed up the convergence, we trained the network in two stages: first we replaced only the convolutional layers with TT-conv layers and trained the network; then we replaced the fully-connected layers with randomly initialized TT-fc layers and fine-tuned the whole model.

To compare against [1] we include the results of compressing only fully-connected layers (Tbl. 1b). Initially, the fully-connected part was the memory bottleneck and it was more fruitful to compress it while leaving the convolutions untouched. But after the first gains, the bottleneck moved to the convolutional part and the fully-connected layers compression capped at about $21\times$ network compression. At this point, by additionally factorizing the convolutions we raised the network compression up to $80\times$ while losing 1.1% of accuracy (Tbl. 1b).

7 Conclusion

In this paper, we proposed a tensor decomposition approach to compressing convolutional layers of a neural network. By combing this convolutional approach with the work [1] for fully-connected layers, we compressed a convolutional network $80\times$ times. These results make a step towards the era of embedding compressed models into smartphones to allow them constantly look and listen to its surroundings. In the future work, we will experiment with the proposed approach on the ILSVRC-2012 dataset [24] on state-of-the-art neural architectures.

References

[1] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 442–450. 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

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

[6] I. V. Oseledets. Tensor-Train decomposition. *SIAM J. Scientific Computing*, 33(5):2295–2317, 2011.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, et al. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[8] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, 2015.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pages 2285–2294, 2015.

[11] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369, 2013.

[12] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *International Conference of Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6655–6659, 2013.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, et al. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[15] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[16] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 2148–2156. 2013.

[17] M. Figurnov, A. Ibraimova, D. Vetrov, and P. Kohli. PerforatedCNNs: Acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems 29 (NIPS)*. 2016.

[18] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[19] M. Wang, B. Liu, and H. Foroosh. Factorized convolutional neural networks. *arXiv preprint arXiv:1608.04337*, 2016.

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

[21] Tensornet experimental codebase `https://github.com/timgaripov/TensorNet-TF`.

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 2015.