Compressing neural networks through CP-decomposition

Théo Rudkiewicz

March 7, 2023

Abstract

Today neural networks are state of the art for numerous task including image classification. The best neural networks use an enormous amount of parameters. Some works try to reduce this number of parameters to be able to make those networks faster. A possible solution to compress neural networks is to decompose the weights tensors of the layers in series of low rank tensors.

1 Introduction

For ten years, neural networks are the best performing algorithms for a large variety of tasks. Among them image classification is probably the most famous. Those neural networks are characterized by an important number of parameters without explicit regularization (for most of the cases) in contrast with traditional algorithms (such as nearest neighbors, binary tree...). For example, AlexNet [KSH17], VGG-16 [SZ14] or ResNet have tens of millions trainable parameters.

This huge number of parameters can lead to interesting questions such as: why is there no irretrievable over-fitting or are all those parameters useful. Many works [CWL+18] have shown that there is redundant information and how to remove it. Different techniques exists such as quantization, pruning or low-rank factorization.

Quantization proposes to lower the numerical precision of weights during storage and computation. Pruning is suppressing some connections between layers, it can be arbitrary connections or blocks of connections to make the network effectively faster. Low-rank factorization compresses the information of the weights tensors representing connections between layers. The low rank factorization can lead to less information and simpler operations.

Those techniques can be either performed during or after training. In the latter case, this allows to compress already trained networks that were

not specifically designed to be compressed. However afterwards compressed networks need sometimes to be fine-tuned.

An important technical point is that when compressing neural networks, it is important to understand how the operations are performed on the appropriate hardware *ie* on GPU. Indeed, in some cases even with a significant reduction of the number of operations the network will not be faster.

2 Notations

I try to use calligraphic letters, like $\mathcal{X}, \mathcal{Y}, \mathcal{W}$ for tensor with order higher than 2. I access to the elements of a tensor with the numpy convention ie $\mathcal{X}[1, 2, 3]$ is the element in position $(1, 2, 3)$. Indices will not be used to access elements, but families or sequences.

The convolution will be from an input $\mathcal{X}$ of size $S \times H \times W$ to a tensor $\mathcal{Y}$ of size $T \times H' \times W'$ with a kernel $\mathcal{K}$ of size $T \times S \times W_d \times H_d$ where $W_d = 2w_d + 1$ and $H_d = 2h_d + 1$. To iterate over S I use s, t over T, x over W, h over H_d, y over H, w over W_d.

A general tensor $\mathcal{X}$ will be of size $I_1 \times \cdots \times I_N$, its decomposition will be of rank R or $R_1, \ldots, R_N$ and the components of the decomposition will be denoted with U (even is U if of order higher than 2). To iter over N I use k, r over R and r_i over R_i.

3 Convolution layer

The convolution layer is one of the most common layer in neural network especially for images. According to [CWL+18] they represent more than 90% of the computation time in AlexNet, VGG or ResNet.

A convolution transforms a tensor $\mathcal{X} \in \mathbb{R}^{S \times W \times H}$ (for example an image of size $H \times W$ with $S = 3$ colors) into an other tensor $\mathcal{Y} \in \mathbb{R}^{T \times W' \times H'}$.

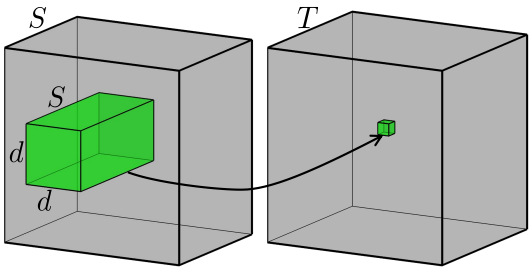

Figure 1: A convolution, here $H_d = W_d = d$[1]

[1] figure from [LGR+15]

The convolution is parameterized by a kernel as which is a dimensional tensor $\mathcal{K} \in \mathbb{R}^{T \times S \times (2w_d+1) \times (2h_d+1)}$. We compute the output $\mathcal{Y}$ with this formula:

$$\mathcal{Y}[x, y, t] = \sum_{s=1}^{S} \sum_{h=-h_d}^{h_d} \sum_{w=-w_d}^{w_d} \mathcal{K}[t, s, y+h, x+w] \mathcal{X}[s, y+h, x+w]$$

For simplicity, in this formula we ignore the special cases at the boundary. We have here a tensor $\mathcal{K}$ that we could decompose to compress in number of parameters and accelerate the convolution operation.

4 Tensor decomposition

4.1 Matrix decomposition

For a matrix $M \in \mathbb{R}^{m \times n}$ the singular value decomposition gives $U \in O(m \times m), \Sigma \in D(m \times n), V \in O(n \times n)$ (where O are the orthogonal matrices and D the diagonal matrices) such that $U\Sigma V = M$. The diagonal of Σ is $(\sigma_1, \ldots, \sigma_k)$, the singular values of M. There are $\operatorname{rank} M$ non zeros singular values. Hence we can write, with U_i the i-th column of U and V_i the i-th line of V:

$$M = \sum_{i=1}^{\operatorname{rank} M} \sigma_i U_i V_i$$

We have a decomposition of M as a sum of low rank matrices (because all the U_i, V_i are of rank 1). If the rank of M is small enough it is a form of compression of M without loss.

Furthermore, we can choose to compress M with a small loss of information. We now assume that the singular values are sorted decreasingly such that $\sigma_{\operatorname{rank} M}$ is the smallest. To compress M with a small loss of information we can choose to neglect the small singular values and keep only the r largest singular values. We then get:

$$\tilde{M} = \sum_{i=1}^{r} \sigma_i U_i V_i$$

$\tilde{M}$ is a good approximation of M because [2]

$$||\tilde{M} - M||_2^2 = \sum_{i=r+1}^{\operatorname{rank} M} \sigma_i^2$$

and we know that the last singular values are usually small.

[2] The use of the Frobenius norm to justify the precision of the approximation is debatable. Especially in the context of compressing neural networks. [DZB+14]

4.2 CP-decomposition

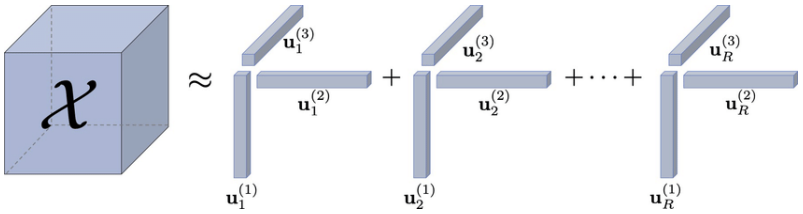

Figure 2: CP-decomposition[3]

The analogous of SVD for tensors is the CP-decomposition [Hit27]. For a tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the CP-decomposition is:

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r U_r^{(1)} \otimes \cdots \otimes U_r^{(N)} \tag{1}$$

Where $\lambda_i \in \mathbb{R}$ is the equivalent of singular value and $U_i^{(j)} \in \mathbb{R}^{n_i}$ is the equivalent of an eigenvectors and R is the rank of the decomposition. An $\otimes$ is the outer product which is the generalisation to tensor of the product of a column vector by a line vector. Hence if we write the full expression of equation (1) we have:

$$\mathcal{X}[r_1, \ldots, k, \ldots, r_N] = \sum_{r=1}^{R} \lambda_r \prod_{k=1}^{N} U_r^{(k)}[i_k] \tag{2}$$

The rank of $\mathcal{X}$ is defined as the smallest rank allowing a loss free decomposition. However in practice the computation of this rank is an NP-hard problem [Hås90] and R is chosen to have a reasonable trade-off between the accuracy of the decomposition and the number of parameters.

4.3 Other decomposition

There exist other possible decompositions like the Tucker decomposition [Tuc66], the tensor train format [Ose11] or the block term decomposition [DL09]. Each of these decompositions leads to different compression techniques [KPY+16, NPOV15, YWL+18] but we will focus on the CP-decomposition.

[3] figure from [PKC+21]

5 Convolution decomposition

5.1 CP-convolution

Now we will use the CP-decomposition to decompose the convolution. We recall that the expression of the convolution is:

$$\mathcal{Y}[t, y, x] = \sum_{s=1}^{S} \sum_{h=-h_d}^{h_d} \sum_{w=-w_d}^{w_d} \mathcal{K}[t, s, h, w]\mathcal{X}[s, y + h, x + w] \tag{3}$$

The tensor $\mathcal{K}$ can be decomposed to a CP-decomposition of rank R: [4]:

$$\mathcal{K} = \sum_{r=1}^{R} U_r^{(T)} \otimes U_r^{(W)} \otimes U_r^{(H)} \otimes U_r^{(S)}$$

$$\mathcal{K}[t, s, h, w] = \sum_{r=1}^{R} U_r^{(T)}[t] U_r^{(W)}[w] U_r^{(H)}[h] U_r^{(S)}[s] \tag{4}$$

[LGR+15] proposes to replace in the expression $\mathcal{K}$ by its CP-decomposition (substitute equation (4) in equation (3)) to get this formula:

$$\mathcal{Y}[t, y, x] = \sum_{r=1}^{R} \sum_{w=-w_d}^{w_d} \sum_{h=-h_d}^{h_d} \sum_{s=1}^{S} U_r^{(T)}[t] U_r^{(W)}[w] U_r^{(H)}[h] U_r^{(S)}[s] \mathcal{X}[s, y + h, x + w]$$

(5)

$$= \sum_{r=1}^{R} U_r^{(T)}[t] \underbrace{\underbrace{\left[\sum_{w=-w_d}^{w_d} U_r^{(W)}[w] \underbrace{\left(\sum_{h=-h_d}^{h_d} U_r^{(H)}[h] \underbrace{\left[\sum_{s=1}^{S} U_r^{(S)}[s] \mathcal{X}[s, y + h, x + w] \right]}_{\text{1×1 conv}} \right)}_{\text{depthwise conv}} \right]}_{\text{depthwise conv}}}_{\text{1×1 convolution}}$$

(6)

[4] Compare to the CP-definition we just put the λ in the first vector *ie* $U_r^{(T)} \leftarrow \lambda_r U_r^{(1)}$.

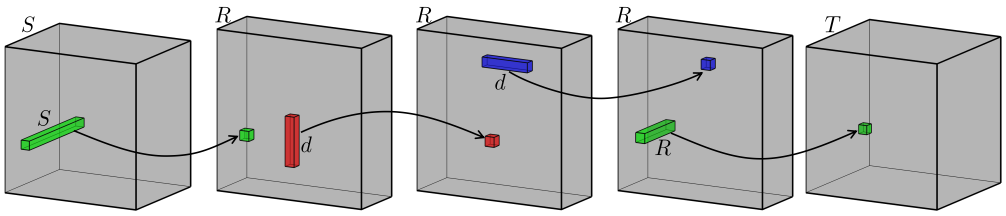

Figure 3: CP-convolution, here $2h_d+1 = 2w_d+1 = d$. The last convolution corresponds to the black sum.[5]

This formula suggest that we can replace one convolution with 4 one dimensional convolution, as illustrated in figure 3.

6 Fine-tuning

Before the decomposition and compression we have a fully trained kernel $\mathcal{K}$. After the decomposition we have an altered $\tilde{\mathcal{K}}$. It is then possible to slightly retrain the model *ie* fine-tune it to recover a better accuracy and a better kernel $\bar{\mathcal{K}}$.

Unfortunately [LGR+15] reported difficulties to retrain the model. This can be due to the fact that the perturbation generated by the compression is too important to recover from it.

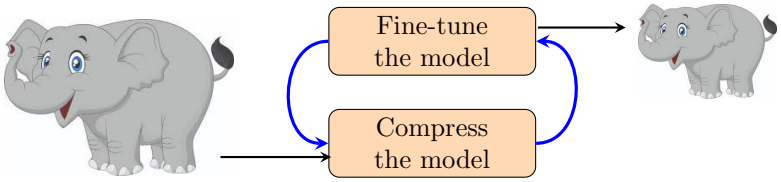

Figure 4: Principle of MUSCO: multiple successive steps of compression and fine tuning[6]

To fix this issue [GKP+19] proposed MUSCO: Multi stage compression, that they summarize with figure 4. To avoid getting too far away from the local optimum they propose to compress with a small compression rate and (almost) recover the local optimum with fine tuning. It is then possible to iterate this procedure until finding a sufficient compression rate.

[5] figure from [LGR+15]
[6] figure from [GKP+19]

7 Obvious Benefits

The benefits are at least a lighter network in terms of size and a faster network in terms of number of operations. In this section I define $W_d = 2w_d + 1$ and $H_d = 2h_d + 1$.

7.0.1 Weight comparison

The initial kernel $\mathcal{K} \in \mathbb{R}^{T \times S \times W_d \times H_d}$ has $T \times S \times W_d \times H_d$ parameters. In comparison, the decomposition $\mathcal{K} = \sum_{r=1}^{R} U_r^{(T)} \otimes U_r^{(W)} \otimes U_r^{(H)} \otimes U_r^{(S)}$ has for each value of r, $T + W_d + H_d + S$ parameters. In total there is $R(T + W_d + H_d + S)$ for the decomposed network. We can conclude that if we have R such that:

$$R < \frac{T \times S \times W_d \times H_d}{T + W_d + H_d + S} \tag{7}$$

then we have compressed the kernel size (or number of parameters) of a factor $\dfrac{T \times S \times W_d \times H_d}{R(T + W_d + H_d + S)}$.

7.0.2 Number of operation comparison

Here we count the number of operation for one pixel with all his channels. For example, in the input there are $H \times W$ pixel with S channels.

For each pixel we can count the number of operation in equation (3), for each pixel there are $S \times W_d \times H_d$ multiplications by new channel which gives $T \times S \times W_d \times H_d$ multiplications for each pixel.

In comparison, as we can count on figure 3, the first convolution needs $S \times R$ multiplications by pixel. Then there are respectively $W_d \times R$, $H_d \times R$ and $R \times T$ multiplications by pixel for the next convolutions. This leads to a total of $R(T + W_d + H_d + S)$ multiplications for each pixel.

We conclude that under the same condition as previously expressed in equation (7) we reduce the number of multiplications by a factor $\frac{T \times S \times W_d \times H_d}{R(T + W_d + H_d + S)}$.

However, a diminution of the theoretical number of operations does not necessarily mean an actual speed-up when the computation is made because the execution time is not purely proportional to the number of operation. Here we have reduced the number of operations by substituting a convolution with 4 smaller convolutions. In this case, as reported in [GKP+19] there is an effective speed-up.

Model	Δ-top-5 accuracy	Compression	Speed-up
AlexNet[7]	-0.81	4.90	2.11
VGG-16[8]	-0.15	1.51	2.41

Table 1: Some results of MUSCO (the Δ-top-5 is the difference of accuracy with the original mode, compression an sped-up are ratio compared to the original model, the speed-up is measured on GPU)

8 Experimental results

In this section I report the results from [GKP+19] because they compressed all the convolution layers (on the contrary [LGR+15] only compressed one layer) and the code is publicly available.

In their experiments they used a generalization of the CP-decomposition that is the Tucker decomposition but the principle is the same. I present the result to give an idea of the efficiency of those low-ranks techniques: it is possible to get a two times faster network with minimal accuracy loss. See table 1 for the results.

9 Bias-variance interpretation and potential hidden benefits

In this section we only consider networks for classification in supervised learning.

In this section, we assume that the data are from a distribution P and $P(y|x)$ give the the probability of the data x to be of the class y. We consider that the goal is to minimize the expected risk for a loss function l: $E_P(f) = \mathbb{E}_{(x,y)\sim P}[l(f(x), y)]$ (we will only write $E(f)$).

9.1 Summary of the error decomposition

The goal of the machine learning algorithm is to find[9] $f^* = \operatorname{argmin}_f E(f)$ We will note $\tilde{f}$ the approximation of f^* found by the algorithm with the available data.

Following the work of [BB07], we can decompose the error in three parts:

1. First the model is constrained with the class of function it can learn. We note $\mathcal{F}$ the ensemble of functions that the algorithm can learn.

[7] [KSH17]
[8] [SZ14]
[9] For all argmin we just suppose that it exists and we choose one of the minimizer.

For example, for a linear model the class of possible functions is quite restrictive, and the possible propositions of the algorithm are only linear functions. Here, for a neural networks, even if the the class of possible functions is bigger we can still have an optimal function that is not in this class of functions. We note $f_{\mathcal{F}} = \text{argmin}_{f \in \mathcal{F}} E(f)$. We therefore have a bias-error or approximation error: $\varepsilon_{app} = E(f^*) - E(f_{\mathcal{F}})$.

2. In addition, we have no access to the theoretical distribution P but only to n samples, giving an approximate distribution P_n. As a consequence we minimize the expected risk with respect to P_n instead of P. This risk is known as the empirical risk: $E_{P_n}(f) = \mathbb{E}_{(x,y)\sim P_n}[l(f(x), y)]$. We can only minimize E_{P_n} with f inside the class $\mathcal{F}$ so we get: $f_n = \text{argmin}_{f \in \mathcal{F}} E_n(f)$ We therefore have a variance-error or estimation error[10]: $\varepsilon_{est} = E(f_{\mathcal{F}}) - E(f_n)$.

3. Even with good optimization techniques, it is not always easy to find a function minimizing a formula. Hence our algorithm only gives an approximation $\widetilde{f_n}$ of f_n. We have therefore an optimization error[11]: $\varepsilon_{opt} = E(f_n) - E(\widetilde{f_n})$.

We can then write the total error as:

$$\varepsilon = \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt} \tag{8}$$

9.2 Impact of the compression on the errors

In this part I discuss the evolution of the different errors during the compression. I write ε_x^c for the errors of the compressed network before fine tuning and ε_x^f for the final errors of the compressed network.

The first step of the compression is the decomposition. The decomposition is a restriction of the class $\mathcal{F}$ because we restrict the rank of the tensors. We note $\widetilde{\mathcal{F}}$ the new class of functions. As $\widetilde{\mathcal{F}} \subset \mathcal{F}$ we have that $\varepsilon_{app}^f = \varepsilon_{app}^c \geq \varepsilon_{app}$.

After the decomposition a step of fine-tuning is performed to recover a better accuracy. As the approximation and estimation errors are fixed, it means that the optimization error is reduced: $\varepsilon_{opt}^f < \varepsilon_{opt}^c$. Moreover as we performed compression by small step to be able to recover the local optimum we can guess that $\varepsilon_{opt}^f \approx \varepsilon_{opt}$. This can be true only if the the added rank constraints don't force to go to a totally different local minima, which could be the case for high compression ratio.

[10] In the original article they consider that the available data is not deterministic and they consider $\varepsilon_{est} = \mathbb{E}(E(f_{\mathcal{F}}) - E(f_n))$ with the expectation taken with respect to the random choice of training set. Here for simplification we do not consider this refinement.

[11] Here we consider that the optimization algorithm is deterministic.

It is more difficult to know how the estimation error changes. In theory, as we increase the bias error we could expect the estimation error to be lower. But in our case, it is special because somehow the approximation error is not independent of the data as we use the data to select the class of possible functions. (Of course it is always a bit the case but it is usually chosen by an expert using prior knowledge and not learned by an algorithm.) As we carefully select $\widetilde{\mathcal{F}}$ we expect the approximation error not to grow too much, therefore if the estimation error was reduced we could expect to find better results for the compressed network, which is not the case.

However, one advantage of some simpler models (than deep neural networks) is their robustness.

9.3 Robustness

After the compression we can't really see a reduction of the estimation error: $\varepsilon_{est} = E(f_{\mathcal{F}}) - E(f_n)$. However rank constraints can be considered as a regularization technique, as the information in the model is constrained to be smaller. As pointed by [JG18, BMCM19] regularization can improve robustness.

Hence even if $E_P[f_{\mathcal{F}}] - E_P[f_n]$ is not reduced, if instead of P we consider $\widehat{P}$ the distribution P with attacks, it is possible that $E_{\widehat{P}}[f_{\mathcal{F}}] - E_{\widehat{P}}[f_n]$ is smaller after the compression than before. We can conclude that robustness could be an hidden benefit of the compression. In fact, tensor decomposition is already used in combination with dropout to improve robustness [BKB$^+$21, KKP$^+$21].

9.4 Transfer learning

An interesting question about compression is how specific is the architecture selection. It is well known that to train a neural network on a task it is easier to start from an other pretrained network on a similar task and then fine-tune it. This allows to reduce the estimation and optimization error because the knowledge given by the first training is reused.

Therefore if the selected architecture by compression (*ie* the rank constraint weight tensors given after a first training) is general enough, it could be even more efficient for transfer learning as there are less weights to update and the network is more regularized. On the contrary, if the model selection is too specific, it will be impossible to learn a new task starting from a decomposed network. To my knowledge, there is at this day no work exploring this possibility.

10 Conclusion

Deep neural networks are known to be over-parameterized. Hence it is possible to compress them without too much loss in precision. One of the possible compression techniques is tensor decomposition. I presented the CP-decomposition which allows compression and speed-up of the network. Moreover, compression is a form of regularization which could lead to other benefits like robustness or easier transfer learning. However those aspects are for now largely unexplored.

References

[BB07] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

[BKB+21] Adrian Bulat, Jean Kossaifi, Sourav Bhattacharya, Yannis Panagakis, Timothy Hospedales, Georgios Tzimiropoulos, Nicholas D. Lane, and Maja Pantic. Defensive Tensorization, 2021.

[BMCM19] Alberto Bietti, Grégoire Mialon, Dexiong Chen, and Julien Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning*, pages 664–674. PMLR, 2019.

[CWL+18] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19:64–77, 2018.

[DL09] Lieven De Lathauwer. A survey of tensor methods. In *2009 IEEE International Symposium on Circuits and Systems*, pages 2773–2776. IEEE, 2009.

[DZB+14] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GKP+19] Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Ivan Oseledets, and Andrzej Cichocki. MUSCO: Multi-Stage Compression of neural networks, November 2019.

[Hås90] Johan Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4):644–654, 1990.

[Hit27] Frank L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[JG18] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 514–529, 2018.

[KKP+21] Arinbjörn Kolbeinsson, Jean Kossaifi, Yannis Panagakis, Adrian Bulat, Animashree Anandkumar, Ioanna Tzoulaki, and Paul M. Matthews. Tensor dropout for robust learning. *IEEE Journal of Selected Topics in Signal Processing*, 15(3):630–640, 2021.

[KPY+16] 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, February 2016.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[LGR+15] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up Convolutional Neural Networks Using Finetuned CP-Decomposition, April 2015.

[NPOV15] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing Neural Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[Ose11] I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[PKC+21] Yannis Panagakis, Jean Kossaifi, Grigorios Chrysos, James Oldfield, Mihalis Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. *Tensor Methods in Computer Vision and Deep Learning.* July 2021.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tuc66] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, September 1966.

[YWL+18] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9378–9387, 2018.