Adaptive Tensor-Train Decomposition for Neural Network Compression

Yanwei Zheng[1], Yang Zhou[2], Zengrui Zhao[1], and Dongxiao Yu[1](✉)

[1] School of Computer Science and Technology, Shandong University, Qingdao 266237, People's Republic of China
{zhengyw,dxyu}@sdu.edu.cn, zhaozr@mail.sdu.edu.cn

[2] Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen 518055, People's Republic of China
zhouyangcumt@163.com

Abstract. It could be of great difficulty and cost to directly apply complex deep neural network to mobile devices with limited computing and endurance abilities. This paper aims to solve such problem through improving the compactness of the model and efficiency of computing. On the basis of MobileNet, a mainstream lightweight neural network, we proposed an Adaptive Tensor-Train Decomposition (ATTD) algorithm to solve the cumbersome problem of finding optimal decomposition rank. For its non-obviousness in the forward acceleration of GPU side, our strategy of choosing to use lower decomposition dimensions and moderate decomposition rank, and the using of dynamic programming, have effectively reduced the number of parameters and amount of computation. And then, we have also set up a real-time target network for mobile devices. With the support of sufficient amount of experiment results, the method proposed in this paper can greatly reduce the number of parameters and amount of computation, improving the model's speed in deducing on mobile devices.

Keywords: Tensor decomposition · Parameter compression · Quantization · Mobile target detection

1 Introduction

In recent years, deep convolution neural network has been widely applied to the computer science including image identification, natural language processing and speech recognition [1]. It has made significant breakthrough in solving various tasks. For example, AlexNet [2] achieved an accuracy in classification 8.7% higher than traditional methods in the 2012 ILSVRC [3] competition. With the growing

Y. Zheng and Y. Zhou—Contribute equally to this work and should be considered co-first authors.

This work is partially supported by National Key R&D Program of China with grant No. 2019YFB2102600 and NSFC (No. 61971269, 61832012).

application of VGGNet [4], GoogLeNet [5] and ResNet [6], artificial intelligence has caught up with and even surpassed human intelligence in classification of massive image data, which can be seen from the fact that the error rate of top-5 classification for ILSVRC has been as low as 3.5% while that for human eyes is about 5.1%.

In order to improve the performance of neural network models, researchers generally design deeper and more complex networks [7]. Deeper networks will greatly increase the number of parameters and amount of computation, and will thus make higher demands on hardware resources (CPU, GPU memory, and bandwidth). As a result, setting up a deep learning system is quite expensive and the cost has become an obstacle for deep neural networks to deal with assignments with limited computing resource or high real-time requirement.

To deploy large-scale convolutional neural network on edge devices, the problem of limited memory space and computation ability needs to be solved. Studies [8] have shown that there are a large number of redundant structures and parameters in the convolutional neural network, especially in the fully connected (FC) layer. The redundant parameters contribute little to the final result, so the network structure and parameters can be compressed to reduce the model size and speed up the computation.

There are five methods to compress and accelerate deep neural network. (1) Parameter pruning, which finds the redundant neurons and removes them [9]. (2) Parameter sharing, which maps multiple parameters with high accuracy to a single parameter with low accuracy using a certain rule [10]. (3) Low-rank decomposition, which decomposes the large matrix into the product of several approximate kernel matrices to reduce the computation [11]. (4) Designing compact convolutional filters, which reduces the computation and parameters of convolution by redesigning the operation steps or methods of convolution kernel [12]. (5) Knowledge distillation, which transfers knowledge from large network to compact distillation model [13,14].

In this paper, we focus on the low-rank decomposition. In 2013, Denil [15] et al. analyzed the effectiveness of low rank decomposition in solving the redundancy problem of deep neural networks. Jaderberg et al. [16] uses the low rank decomposition technique of tensor to decompose the original convolution kernel into two smaller convolution kernels. Using the classical tensor decomposition algorithm CP [17], the parameter tensor can be decomposed into the sum of several smaller rank one matrices. Using the Tucker decomposition [18], the parameter tensor can be decomposed into the product of a core tensor and several smaller tensors. Using Tensor-Train decomposition [19], the original parameter tensor can be decomposed into the product of multiple matrices, as shown in Fig. 1. Compared with CP and tucker decomposition, Tensor-Train decomposition has a more compact structure, and the representation of matrix multiplication makes the compressed tensor easier to operate.

At present, most of the rank of Tensor-Train decomposition is set by adjusting parameters empirically. The decomposition ranks of each layer need to be determined manually, and there are many layers in a neural network. It is

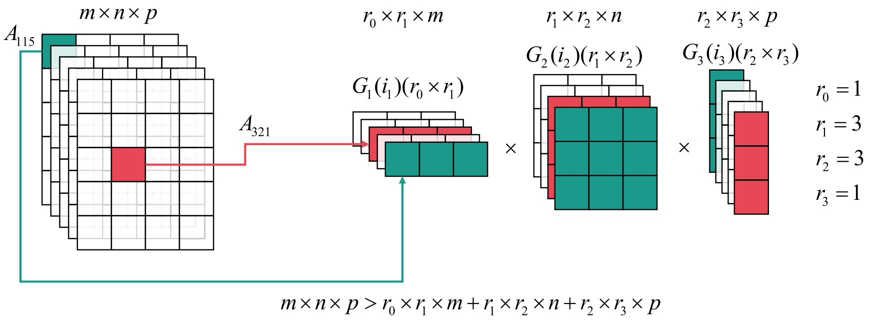
$m \times n \times p$
A_{115}
A_{321}
$r_0 \times r_1 \times m$
$G_1(i_1)(r_0 \times r_1)$
$r_1 \times r_2 \times n$
$G_2(i_2)(r_1 \times r_2)$
$r_2 \times r_3 \times p$
$G_3(i_3)(r_2 \times r_3)$
$\times$
$\times$
$r_0 = 1$
$r_1 = 3$
$r_2 = 3$
$r_3 = 1$
$m \times n \times p > r_0 \times r_1 \times m + r_1 \times r_2 \times n + r_2 \times r_3 \times p$

Fig. 1. Tensor-Train Decomposition. A 3D $m \times n \times p$ matrix is approximately decomposed into some small into the product of several small matrices. The sum of calculation amount of the small matrices is less than that of the original matrix.

difficult to achieve low precision loss and high compression ratio at the same time. This paper proposes an Adaptive Tensor-Train Decomposition (ATTD) method. After presetting a precision and the decomposition dimension, singular value decomposition (SVD) is used to directly calculate the optimal decomposition rank. Only one parameter – the precision – needs to be adjusted in the whole process. We summarize our contributions as follows:

- Based on the work of Tensorizing Neural Networks (TNN) [20], we propose an ATTD algorithm which can adaptively compute the optimal decomposition rank of each layer of network. Compared with the manual adjustment method, this method leads to the smaller accuracy loss and the larger compression ratio.
- We use the improved ATTD algorithm to further compress the depthwise separable convolution that is widely used in the current mainstream lightweight networks.
- By combining the compression and acceleration strategy with the quantitative algorithm, a lightweight target detection network based on MobileNet is built on mobile devices. It has nearly doubled the model acceleration effect.

The remainder of this paper is organized as follows. Section 2 discusses the related work about the neural network compression. In Sect. 3, the ATTD model is presented. Section 4 provides the experimental results. Section 5 concludes this paper and outlines the future work.

2 Related Work

This section discusses the compression and acceleration methods in deep neural networks.

Parameter Pruning. Parameter pruning reduces the amount of model parameters by deleting redundant parameters in neural network. For unstructured pruning, [9,21,22] all reduced the amount of network parameters while ensuring a certain accuracy. In order to solve the problem that unstructured pruning would cause a large number of unstructured sparse connectivities, researchers have proposed methods based on structured pruning [23–25], which, while directly compressing the neural network, have effectively accelerated the computing speed of the entire model.

Parameter Sharing. Parameter sharing aims to map multiple parameters with high accuracy to a single parameter with low accuracy using a certain rule. Parameter quantization uses low-precision data type to replace the original 32-bit full-precision floating-point [26–28]. The binarization method uses binary weights to represent the parameters or activation functions of the model [29–31]. Methods such as hash function [15] and structured linear mapping [16] enable the parameter of the FC layer to be shared, which significantly reduces the memory needed for neural network.

Low-Rank Decomposition. Applying matrix or tensor decomposition algorithms to convolutional and FC layers can compress and accelerate deep neural networks [32]. The classical tensor decomposition algorithms such as CP [33] decomposition, Tucker [18] decomposition, and Tensor-Train [19] decomposition all decompose the original parameter tensor with low rank. In [17], the low rank filter with rank 1 is constructed by cross channel and filter redundancy. [34] finds the exact global optimizer of the decomposition, and proposes a method for training low-rank constrained Convolution Neural Networks (CNNs) from scratch. [35] attempts to reduce spatial and channel redundancy directly from the visual input for CNN acceleration.

Designing Compact Convolutional Filters. SqueezeNet [36] replaces the original convolution structure with the Fire Module structure, which can significantly compress the model parameters. Google's MobileNet [37] replaces the original convolution with depthwise separable convolution which is of a smaller amount of computation and parameters; ShuffleNet [38] uses group convolution and channel shuffle to design a new convolution structure. Both have reduced the amount of model parameters and computations.

Knowledge Distillation. The mean idea of knowledge distillation is to distillate certain knowledge from a complex teacher model to a simpler student model. The compression and acceleration strategy based on knowledge distillation [13,14,39,40] can convert knowledge of large-scale networks to small-scale ones, which effectively reduces the amount of computation than the original network.

3 The Proposed Method

3.1 Compress Traditional Network by TT Decomposition

High-order tensors in practice are generally sparse, and direct operations between tensors will waste a lot of computing resources. Decomposition of high-order tensors can effectively reduce the amount of computation. The principle of Tensor-

Train [19] decomposition is to represent each element in a high-dimensional tensor as a matrix multiplication. The Tensor-Train decomposition form of tensor A can be written as:

$$A(i_1, i_2, ..., i_d) = G_1(i_1)G_2(i_2)...G_d(i_d), \tag{1}$$

where $G_k(i_k)$ is a matrix of $r_{k-1} \times r_k$, and the dimension of tensor A is d, so the number of matrices obtained by decomposition is also d. The result of multiplying several matrices represents an element in tensor A. To ensure that the final result is a scalar, we set $r_0 = r_d = 1$.

Figure 1 shows the Tensor-Train decomposition process of a 3D tensor. Any element in tensor A, like A_{321}, can be written in the form of continuous multiplication of 3 matrices. Here, the decomposition rank of the Tensor-Train is set to $(1, 3, 3, 1)$, and the size of each matrix is $r_{k-1} \times r_k$, which of the example mentioned before are 1×3, 3×3, 3×1, respectively. The position of each matrix in G_k is determined by the element's subscript $i_k \in [1, n_k]$, which is $3, 2$, and 1, respectively. The original tensor has a total of $5 \times 4 \times 5 = 100$ parameters, while after compression, there are a total of $1 \times 3 \times 5 + 3 \times 3 \times 4 + 3 \times 1 \times 5 = 66$ parameters.

3.2 Adaptive Tensor-Train Decomposition

The decomposition algorithm based on Tensor-Train is able to significantly compress the parameters of the FC layers and convolutional layers, but practically, the Tensor-Train decomposition rank $r_1, ..., r_{k-1}$ of each layer of the network needs to be set manually ($r_0 = r_1 = 1$). If there are n layers to be compressed, $n(k-1)$ decomposition ranks will have to be adjusted. If the k and n are large, there will be lots of parameters that need to be set manually, and it will be difficult to set the optimal decomposition rank to obtain a relatively high compression rate while ensuring a small loss of accuracy.

The larger singular value will determine the main feature of the original matrix. The Tensor-Train decomposition considers the larger singular value and ignores the smaller singular value, that is, only the main feature is considered and the secondary feature is ignored. Thus, Tensor-Train decomposition precision is positively correlated with the singular value. The larger the singular value, the greater the contribution to the precision. Inspired by this, we define the precision as

$$\varepsilon \approx \frac{\sum_{i=1}^{r_k} \sigma_i}{\sum_{i=1}^{n} \sigma_i}, \tag{2}$$

where ε is the ratio of the selected top r_k singular values to the sum of all singular values, σ_i is the singular value of i, and $\sigma_i \leq \sigma_{i-1}$.

According to Eq. 2, using ATTD algorithm with the preset precision ε, the optimal decomposition rank at a current accuracy can be directly computed according to the pre-trained network after the decomposition dimension d is set, as shown in Algorithm 1 .

Algorithm 1. ATTD Algorithm

```
Input: $A, d, \varepsilon$
Output: $[G_1, ..., G_d]$
 1: $B = A, r_0 = 1, P = \prod_{s=1}^{d} n_s$
 2: for $k = 1$ to $d - 1$ do
 3:     $S_{left} = r_{k-1}n_k$, $S_{right} = \frac{P}{r_{k-1}n_k}$,
        $B = reshape(B, [S_{left}, S_{right}])$
 4:     Apply $\varepsilon - truncated$ SVD on B: $[U_k, \Sigma_k, V_k] = SVD(B)$
 5:     $P = \frac{P}{r_{k-1}n_k} r_k$
 6:     $G_k = reshape(U_k, [r_{k-1}, n_k, r_k])$
 7:     $B = \Sigma_k V_k$
 8: end for
 9: $G_d = B$
10: return TT-Cores $[G_1, ..., G_d]$.
```

The whole process only needs to adjust one parameter of ε. Given the value of ε, how many singular values participate in the decomposition is determined. That is, the decomposition ranks of each layer are determined. We do not have to adjust the decomposition ranks of each layer manually.

3.3 Depthwise Separable Convolution Based on ATTD

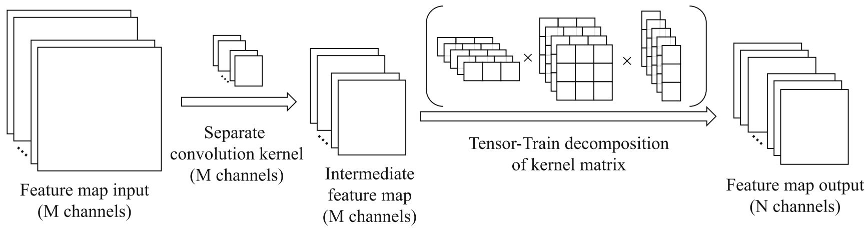

Fig. 2. Tensor-Train Decomposition. A 3D $n_1 \times n_2 \times n_3$ matrix is approximately decomposed into some small into the product of several small matrices. The sum of calculation amount of the small matrices is less than that of the original matrix.

Google proposed depthwise separable convolution in MobileNet, which separates the convolution into two steps: depthwise convolution and pointwise convolution. According to the statistics [37], 95% of the computations and 75% of the parameters in MobileNet come from 1×1 convolution, and the overall distribution of weight parameters roughly conforms to a normal distribution. There are a large number of parameters whose value is around 0, which does not contribute to the network, so 1×1 convolution has a large number of redundant parameters.

Suppose the shape of convolution kernel parameter matrix is $1 \times 1 \times M \times N$, where M and N are the numbers of input and output feature map channels, thus the total parameter of 1×1 convolution is MN. The parameter matrix can be regarded as a FC matrix. We use the adaptive Tensor-Train decomposition algorithm to compress it. The specific steps are:

(i) Transform the convolution kernel matrix of current layer into a tensor A with dimensions $(m_1n_1, ..., m_dn_d)$, where $\prod_{i=1}^{d} m_i = M$, $\prod_{i=1}^{d} n_i = N$.
(ii) Apply the adaptive Tensor-Train decomposition algorithm to A to obtain the kernel matrix $G_k[m_k, n_k]$.
(iii) Decompose M with the same way in step (ii) to obtain matrix $\mathcal{X}(x, y, m_1, ..., m_d)$, where $\prod_{i=1}^{d} m_i = M$. After tensor operation, we obtain the output feature map $\mathcal{Y}(x, y, n_1, ..., n_d)$, where $\prod_{i=1}^{d} n_i = N$.

The operation process of 1×1 convolution can be shown as:

$$\mathcal{Y}(x, y, n_1, ..., n_d) = \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{m_1,...,m_d} \mathcal{X}(x, y, m_1, ..., m_d)G_0G_1[m_1, n_1]...G_d[m_d, n_d] \quad (3)$$

The depth separable convolution after the introduction of Tensor-Train decomposition is shown in Fig. 2.

3.4 Improvement of Inference Speed and Optimization of GPU

After the addition of the adaptive Tensor-Train decomposition module, the actual inference speed is not significantly improved compared to the original model although the amount of parameters and computation of the model has significantly reduced. The main reasons are as follows:

(i) Tensor-Train decomposition decomposes a large parameter matrix into several compact 3D tensor forms, also known as matrix product states. Such small tensor operations cannot effectively use the GPU's parallel computing capabilities for large matrices.
(ii) The adaptive Tensor-Train decomposition algorithm tends to find a decomposition method with a higher decomposition dimension, which will cause increasing small 3D tensors after decomposition, and a larger decomposition rank, which will lead to parameter redundancy.
(iii) The order of computing irregular matrix after decomposition will affect the final computation amount theoretically. The difference in computation amount between the worst and the best order varies from several times to several tens of times.

For problems (i) and (ii), we adopt a strategy of low decomposition dimension and moderate decomposition rank. For problem (iii), after training the model, we use dynamic programming to compute the optimal matrix operation order of each layer and adjust them. The principle of the algorithm is:

(i) Let $A[i : j] = A_iA_{i+1}...A_j$, where $A[i : j]$ denote the product of the i-th matrix to the j-th matrix. For $k(i \leq k < k)$, the computation amount of

$A[i:j]$ is: the sum of there computation amount including the computation amount of $A[i:k]$ and that of $A[k+1:j]$, and the amount of computation multiplied by the two. Let $P[i-1]$ and $P[i]$ be the dimensions of the i-th matrix.

(ii) Let $C[i][j]$ be the amount of computation required for the optimal order of computation of $A[i:j]$, then:

$$C[i][j] = \begin{cases} 0, & if(i == j) \\ min\{C[i][k] + C[k+1][j] + P[i-1]P[k]P[j]\}, & i \leq k < j \end{cases} \tag{4}$$

(iii) Update the cost matrix $C[i][j]$ and the marking matrix $S[i][j] = k$ from the bottom up.

(iv) Return the optimal computation method of $C[i][j]$ according to the separation operation provided by $S[i][j]$.

4 Experiments

4.1 Datasets

CIFAR-10 contains 50,000 training pictures and 10,000 test pictures, both in a total of 10 categories. We use this dataset to verify the adaptive Tensor-Train decomposition algorithm's ability of neural network parameter compression.

COCO dataset is currently the mainstream target detection dataset. We built a target detection network based on our proposed method on mobile devices to verify the feasibility and effectiveness on COCO.

4.2 Implementation Details

On CIFAR-10 dataset, we use the SGD optimizer with $momentum = 0.9$. The batchsize is set to 32, and the initial learning rate is 0.1. After 100 iterations, drop the learning rate 10 times every 20 iterations. In order to verify the compression ability of the algorithm on the FC layer and the convolutional layer, we designed 3 structures in this chapter: the network dominated by the FC layer, the network dominated by the convolutional layer, and a classic network composed of both layers. Use adaptive Tensor-Train decomposition algorithm to compress the above 3 models, where the Tensor-Train decomposition ranks are set manually and initialized randomly. Set ε from 0.7 to 0.98, and set $d = 3$.

On the pre-trained MobileNet V1 and V2 [41] models, we use the adaptive Tensor-Train decomposition algorithm to compress the deep separable convolution. The parameter settings remain unchanged.

4.3 Effectiveness of Proposed Method

Effectiveness on Traditional Network. Table 1 shows the results that Tensor-Train decomposition works well on compressing the FC layers where the

Table 1. Results of accuracy and compression ratio of different network structures

Network	Average acc	Compress rate
FULLY(base)	84.8%	1
FULLY(TT)_1	84.6%	20.32
FULLY(TT)_2	83.9%	43.35
FULLY(TT)_3	82.8%	82.74
CONV(base)	90.2%	1
CONV(TT)_1	88.1%	2.28
CONV(TT)_2	86.9%	2.64
CONV(TT)_3	85.3%	3.72
CONV+FULLY(base)	91.21%	1
CONV+FULLY(TT)	89.01%	43.41
CONV+FULLY(ATTD)	88.94%	65.45

Note: Base represents the uncompressed network, FULLY represents the network dominated by FC layers, CONV represents the network dominated by the convolutional layers, CONV+FULLY represents the classic network composed of FC layers and convolutional layers, TT represents the network compressed by Tensor-Train decomposition algorithm, and ATTD represents the network compressed by ATTD algorithm.

effect reaches up to 83 times, and the accuracy loss of the network is about 2%. The compression effect of this algorithm on the convolutional layer is unsatisfactory with only 2 to 4 times, and the accuracy drops quickly, with the accuracy loss reaching about 5%. This is because the parameters of the convolution layer are far less than that of the fully connected layer, so the compression effect is limited. The compression rate of the classic network can reach 43.41 times, and the accuracy loss is reduced by about 1.2%.

The adaptive Tensor-Train decomposition algorithm surpasses the manual adjustment method in accuracy and compression ratio of the model. Within similar accuracy decline (about 1.3%), the compression rate of adaptive Tensor-Train decomposition algorithm can reach about 65 times, greatly exceeding the result of 43 times of the CONV+FULLY model.

Effectiveness on MobileNet. As Table 2 shows, for MobileNet V1, when the value of ε is 0.9, the parameter amount of the model is reduced by about 3 times, the model accuracy is reduced by about 1.6%, and the theoretical computation (Multi-Adds) is nearly halved, but the improvement of inference speed is insignificant.

For MobileNet V2, when the value of ε is 0.9, the parameter amount of the model is reduced by about 4 times, computation amount is reduced by more than half, and the model accuracy is reduced by about 1.8%, but the inference speed is also not significantly improved compared to the original model.

Table 2. Comparison of V1 and V2 before and after compression when $\varepsilon = 0.9$

Network	Acc	Params	MAdds	FPS	ε	Compr
MobileNet V1(base)	94.23%	4.2M	569M	1053		1
MobileNet V1(ATTD)	92.63%	1.3M	305M	1196	0.9	3.28
MobileNet V1(ATTD_Re)	92.51%	0.9M	181M	**1582**	0.9	4.6
MobileNet V2(base)	95.49%	3.4M	300M	1141		1
MobileNet V2(ATTD)	93.78%	0.78M	141M	1365	0.9	4.32
MobileNet V2(ATTD_Re)	93.72%	0.59M	98M	**1694**	0.9	5.8

Note: The accuracy, parameter amount, computation amount, inference speed and compression rate of MobileNet V1 and V2 when value of ε is 0.9.

Table 3. Comparison of V1 and V2 before and after compression when $\varepsilon = 0.9$

Network	FPS
MobileNet V1(base)	1053
MobileNet V1(ATTD_Re)	1582
MobileNet V1(ATTD_Re_Quan)	**2443**
MobileNet V2(base)	1141
MobileNet V2(ATTD_Re)	1694
MobileNet V2(ATTD_Re_Quan)	**2801**

After fine-tuning the ATTD algorithm and improving the operation order of parameter matrices, the inference speed of the two models has been significantly improved, with FPS reaching 1582 and 1694 respectively, and the amount of parameters has also decreased.

Quantitative technology has a significant effect in acceleration on the inference speed of neural networks. As Table 3 shows, quantization technology enables the model to achieve an inference speed acceleration of about 2 times, which is of great significance for mobile and embedded devices.

5 Conclusion

In this paper, the application of the Tensor-Train decomposition algorithm in model compression and acceleration is reviewed and studied. The algorithm is applied to the mainstream lightweight neural network MobileNet, during which some improvements are made in terms of its advantages and disadvantages. As a result, our method has increased the efficiency of the model and reduced the memory needed. In addition, we have also built a target detection network based on this algorithm on mobile devices, through which we verified the feasibility and effectiveness of our algorithm and strategy on mobile CPUs.

References

1. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS, pp. 1097–1105 (2012)
3. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
4. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014)
5. Szegedy, C.: Going deeper with convolutions. In: CVPR, pp. 1–9 (2015)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778 (2016)
7. Sheng, H., et al.: Mining hard samples globally and efficiently for person reidentification. IoT-J **7**(10), 9611–9622 (2020)
8. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: NIPS, pp. 1135–1143 (2015)
9. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks (2015)
10. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks (2017)
11. Bach, F.R., Jordan, M.I.: Predictive low-rank decomposition for kernel methods. In: ICML, pp. 33–40. Association for Computing Machinery (2005)
12. Prakash, A., Storer, J., Florencio, D., Zhang, C.: Repr: Improved training of convolutional filters (2018)
13. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
14. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets (2014)
15. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: ICML, pp. 2285–2294 (2015)
16. Cheng, Y., Yu, F.X., Feris, R.S., Kumar, S., Choudhary, A., Chang, S.F.: An exploration of parameter redundancy in deep networks with circulant projections. In: ICCV, pp. 2857–2865 (2015)
17. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions (2014)
18. Zhang, J., Han, Y., Jiang, J.: Tucker decomposition-based tensor learning for human action recognition. Multimedia Syst. **22**(3), 343–353 (2015). https://doi.org/10.1007/s00530-015-0464-7
19. Oseledets, I.V.: Tensor-train decomposition. SIAM J. Sci. Comput. **33**(5), 2295–2317 (2011)
20. Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D.P.: Tensorizing neural networks. In: NIPS, pp. 442–450 (2015)
21. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: NIPS, pp. 598–605 (1990)
22. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (2015)
23. Lebedev, V., Lempitsky, V.: Fast convnets using group-wise brain damage. In: CVPR, pp. 2554–2564 (2016)

24. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference (2016)
25. Luo, J.H., Wu, J., Lin, W.: Thinet: a filter level pruning method for deep neural network compression. In: ICCV, pp. 5058–5066 (2017)
26. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: ICML, pp. 1737–1746 (2015)
27. Ma, Y., Suda, N., Cao, Y., Seo, J.S., Vrudhula, S.: Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In: International Conference on Field Programmable Logic and Applications, pp. 1–8 (2016)
28. Gysel, P.: Ristretto: hardware-oriented approximation of convolutional neural networks (2016)
29. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: training deep neural networks with binary weights during propagations. In: NIPS, pp. 3123–3131 (2015)
30. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1 (2016)
31. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
32. Denil, M., Shakibi, B., Dinh, L., Ranzato, M.A., De Freitas, N.: Predicting parameters in deep learning. In: NIPS, pp. 2148–2156 (2013)
33. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned cp-decomposition (2014)
34. Tai, C., Xiao, T., Zhang, Y., Wang, X.: Convolutional neural networks with low-rank regularization (2015)
35. Lin, S., Ji, R., Chen, C., Huang, F.: Espace: accelerating convolutional neural networks via eliminating spatial and channel redundancy. In: AAAI, pp. 1424–1430 (2017)
36. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
37. Howard, A.G.: Efficient convolutional neural networks for mobile vision applications, Mobilenets (2017)
38. Hluchyj, M.G., Karol, M.J.: Shuffle net: an application of generalized perfect shuffles to multihop lightwave networks. J. Lightwave Technol. **9**(10), 1386–1397 (1991)
39. Korattikara, A., Rathod, V., Murphy, K., Welling, M.: Bayesian dark knowledge. In: NIPS, pp. 3438–3446 (2015)
40. Luo, P., Zhu, Z., Liu, Z., Wang, X., Tang, X.: Face model compression by distilling knowledge from neurons. In: AAAI, pp. 3560–3566 (2016)
41. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv 2: inverted residuals and linear bottlenecks. In: CVPR, pp. 4510–4520 (2018)