Incremental Trained Ternary Quantization

Mostafa Pashazadeh
University of Victoria
Computer Science
mostafapashazadeh@uvic.ca

Abstract

1. Introduction

Today's deep learning model has dominated many machine learning applications and has made significant improvement on a variety of computer vision tasks including image classification, face recognition, and object detection. Mathematically deep learning networks aim to estimate an optimum function with the order of millions parameters that accurately model the relation between input data and output task. However, the inference operation requires large computing resources that makes it increasingly difficult to deploy the model on small devices.

substantial efforts including low precision arithmetic [1, 2, 26, 8, 3, 18, 34, 22, 23, 35], low-rank factorization [12, 15, 28, 14, 14], parameter pruning [17, 4, 16, 31, 19, 20, 6, 21], and student-teacher framework [7, 27, 24, 30, 13] have been made to reduce the size of CNNs and accelerate their adoption on small devices. These methods can complement each other. For example, [29] combine network pruning and weight quantization, and [25] jointly leverage weight quantization and knowledge distillation. As a parallel line of work, some other compression methods focus on designing models with few parameters while preserving accuracy [11, 9, 10]. Among existing methods, low precision arithmetic that cover binarization, ternarization, and quantization has attracted great attention from researchers.

[32] proposes incremental network quantization that converts pretrained full-precision CNN model into low-precision version whose weights are constrained to be either powers of two or zero. Thus, it can transform multiplications into inexpensive bitwise shift operations (i.e., $x \times 2^w = x \ll w$) where x is representative activation value and 2^w is representative weight value after quantization. In each round, it divides the weights in each layer of a pre-trained full precision CNN model into two disjoint groups. The weights in the first group are quantized and responsible for forming a low-precision base. The weights in the second group are the ones to be retrained and compensate for the loss in model accuracy. Once the first run of the quantization and retraining operations is finished, the process is further conducted on the second weight group in an iterative manner, until all the weights are converted to be either powers of two or zero. Experimental results show that considering 5-bit low-precision versions of ResNet-50 trained on Imagenet, this method decreases top-1 error by 1.59% against 32-bit floating-point baselines.

While binary precision models [26, 33, 2, 3] benefits from $32\times$ smaller model size, the extreme compression rate comes with a loss of accuracy. [18] proposes ternary weight networks (TWN) that seek to reach a desired balance between model size and accuracy. Ternary model reduces accuracy loss of binary networks by introducing zero as the third quantized value. For each filter f, they use two symmetric thresholds $\pm\triangle_f$ and a scaling factor W_f to quantize weighs into $\{-W_f, 0, W_f\}$. Ternary values are attributed to weights less than the negative threshold, close to zero, and higher than the positive threshold, respectively. They seek to minimize the Euclidian distance between the full precision weights and the ternary valued weights. In this way, it requires only one multiplication per activation. As there is no deterministic optimum solution to the problem, they assume that the weights of original model follow a normal distribution, and finds an approximation solution to the problem. Adopting ResNet-18 on ImageNet dataset, TWN achieves a validation accuracy that is close to full precision networks according to their paper. [34] proposes trained ternary quantization (TTQ) which differently from [18] uses two independent learnable scaling coefficients W_l^p, W_l^n for each layer l, and two symmetric thresholds defined as a factor of the maximum absolute value of layer's weights. Here the threshold factor is a hyper-parameter that is the same across all the layers in order to reduce the search space. Then, they conduct trained quantization by back propagating two gradients, i.e., gradients with respect to full precision weights and scaling coefficients. The former enables learning the ternary assignments, and the latter enables learning the ternary values. Experimental results show that TTQ slightly improves the accuracy of ResNet56 trained on CIFAR-10 dataset.

However, achieving near full precision performance on ImageNet with very deep network is still an ongoing challange. Our work is an attempt to address this problem by engineering existing approaches to further boost the accuracy. With this intuition we introduce the incremental Trained Ternary Quantization (ITTQ) approach that make the following contributions:

- We believe that there is no concrete evidence that filters' weights follow normal distribution, and weights that learn different feature maps are likely to follow different distributions. Diverse weight distribution hypothesis has also been supported by [22]. Furthermore, the range of weights magnitude are most likely to vary across filters within a layer. Consequently, many weights are pruned out (ternarized to zero) if we consider the same threshold for all filters. Also, considering the same pair of learnable scaling factors for all filters gives rise to quantization error and performance degradation. These led us to learn two independent scaling factor for each filter and customize the threshold for each filter as a factor of the maximum absolute value of filter's weight.

- Despite the fact that filter-wise ternary method appears to introduce more parameters, the concept of model acceleration still stay alive. We argue that since each filter is implemented as matrix multiplication, the number of multiplication operation is the same as the layer-wise ternary method in [34].

- We deploy incremental training strategy in line with [32] with the difference that both the full precision weights and ternarized weights are trained. At each round, we set the initial scaling factor as the average value of the weights which are supposed to be ternarized. We hypothesize this strategy will provide good initial parameters and help to avoid getting trapped in poor local optimum. Because we average over more unanimous weights that help to reduce the quantization error and model distortion at the beginning of each round.

- previous network compression methods [26, 33, 3, 18, 34] suffer from performance degradation on deep CNNs trained with massive ImageNet dataset, or does not provide any experimental results in this case. We aim to achieve a robust method capable to be deployed in this circumstances.

2. Incremental Trained Ternary Quantization

Since convolutional filters account for most of the multiplications in CNNs, we focus our efforts to speed up this filter. In this section, we describe key components of our ITTQ, and detail its implementation.

2.1. Trained Ternary Values

To learn the ternary value, we introduce two scaling factors W_f^p and W_f^n for positive and negative weights in each filter. This follows the same spirit in [34] with the difference that they are tailored to each filter within the convolutional layer. Then, we normalize the full precision weights to the range $[-1, 1]$ by dividing each weight by the maximum absolute value of its filter's weights. Next, we ternarize the full precision weights as follows:

$$w_f^t = \begin{cases} W_f^p : \ w_f > \triangle_f \\ 0 : \ |w_f| \le \triangle_f \\ -W_f^n : \ w_f < -\triangle_f \end{cases} \tag{1}$$

Here, w_f^t is the ternary weight and w_f is the full precision weight. For all filters, we maintain a constant factor t of the maximum absolute value of the weights as the threshold as follows: $\Delta_f = t \times max(|w_f|)$. Finally, we conduct trained ternary quantization by back propagating to both full precision weights and scaling coefficients. The former enables learning the ternary assignments, and the latter enables learning the ternary values. We noticed that the get-through scheme in [26, 33] for computing gradients with respect to full precision weights works slightly better than the scaled gradients adopted in [34]. Based on the chain rule of gradient, derivatives of W_f^p and W_f^n are calculated as $\frac{\partial L}{\partial W_f^p} = \sum_{i \in I_f^p} \frac{\partial L}{\partial w_f^t(i)}$, $\frac{\partial L}{\partial W_f^n} = \sum_{i \in I_f^n} \frac{\partial L}{\partial w_f^t(i)}$. Here $I_f^p = \{i | w_f(i) > \triangle_l\}$ and $I_f^n = \{i | w_f(i) < -\triangle_f\}$. Furthermore, get-through gradients for full precision weights would be $\frac{\partial L}{\partial w_f} = \frac{\partial L}{\partial w_f^t}$. It is noteworthy that thresholds are varying over training iterations since the maximum absolute value changes as training moves forward.

2.2. Incremental Steps

In previous low precision methods [26, 33, 2, 3, 18, 34] all the weights are converted to ternary values at once which in turn causes accuracy loss. Recall that our objective is to achieve a robust and accurate ternary quantization method to reliably accelerate any pre-trained full precision model. Toward this goal, we further incorporate an incremental strategy in line with the work in [32] to steadily convert the full precision weights into ternary values over several rounds. Each round encompasses three interdependent operations: weight partition, ternarizing, and re-training. Weight partition is to divide the weights in each filter of a pre-trained full precision CNN model into two disjoint groups. Also, pruning inspired strategy is used for weight partition i.e., weights with larger absolute values are more important than the smaller ones. Thus, they are responsible for forming a low precision base for the original model and fall in the first group. The rest of the weights adapt to compensate for accuracy loss and fall in the second group. Differently from [32], we train both groups rather than just counting on the weights in the second group to compensate for the model distortion. Here the main objective of incremental technique is to help bridge between subsequent rounds by proving good initialization for the following round. At each round, we set the initial scaling coefficient as the average value of the first group of weights i.e., the weights that are supposed to be ternarized. Full-precision model is more likely to get away from poor local minima while training because of massive redundancy in the model parameters. However, compression techniques considerably eliminate the redundancy. Also, they introduce noisy term into back propagation when updating parameters [35]. We hypothesize this strategy will provide good initial parameters and help to avoid getting trapped in poor local optimum. The rationale is that by each round, part of the parameters are already ternarized and have a unanimous value, as a result, taking the average of the parameter group, that is supposed to be ternarized, as the the initial scaling factors at the beginning of the next round, reduce the quantization error and does not move the model drastically away from the optimum state. Once the first run of the ternarization and re-training operations is finished, all the procedure is further conducted on the second weight group in an iterative manner, until all the weights are converted to ternary values.

3. Experimental Results

For experimental results, we focused on Resnet-101 [5] using ImageNet dataset, to show the efficacy of our method on deep models with massive datasets

3.1. Tiny ImageNet

I examined the incremental filter-wise trained ternary quantization method that make use of two independent learnable scaling coefficients for each set of filter parameters along with incremental strategy. Incremental strategy involves several round of ternarization. During each round, an accumulative proportion of filter parameters is ternarized. In the final round it becomes one and all the filter parameters will be ternarized. This incremental strategy is planned to help bridge between subsequent rounds, that is, to provide good initial learnable scaling factors that alleviate the model distortion at the beginning of each round. The reason is that we compute the initial scaling coeffiicents for each round as the average over more unanimous weights which in turn reduce the quantization error. Actually this hypothesis is somewhat justified by the experimental results that come later in this report, but the problem is that after quite a few number of training iteration we do not achieve competetively acceptable accuracy.

The experiments are first conducted on Tiny-ImageNet dataset rather than ImageNet dataset to examine the efficacy of the method. Tiny-ImageNet has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. However, I realized that the relatively lightened training dataset gives rise to the overfitting problem even though the model is downsized by limiting the parameters to take ternary values. I will go through this problem and possible solution later in this report. Furthermore, it will take too much time for Compute Canada to allocate the required resources to the code if the model is set to be trained on ImageNet because the estimated run time increases considerably. To gain an insight into the functionality of each phase of our method, I sought to measure the efficacy of filer-wise trained ternary quantization and further the incremental filter-wise trained ternary quantization, separately. Also, I compared the results with layer-wise trained ternary quantization method proposed in [34]. Furthermore, I noticed that [18] has developed a filter-wise ternary network, but instead of learning scaling factors, it approximates the distribution of filter parameters as Gaussian and figure out single scaling factor that minimize the Euclidian distance between the full precision weights and the ternary-valued weights i.e., quantization error. This method is implemented and used as the benchmark to see how competitive our results are.

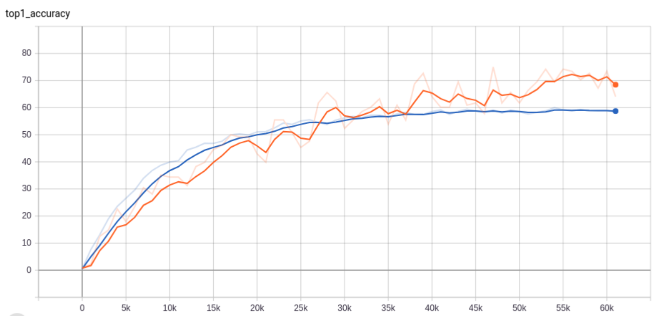
In the following figures, that showcase the experimental results, the red graph relates to training dataset and the blue one relates to validation dataset. The results for full-precision ResNet101 on tiny-ImageNet dataset is shown in Fig1. Although the results are sensible and satisfying to some extent, the model seems too big for this dataset such that we cannot exploit the full capacity of the model. The best hyper-parameters for filter-wise trained ternary model, resulted from exhaustive search in parameters space, are threshold factor = 0.05, learning_rate = 2e-5 for ternarized and full precision parameters, and learning_rate = 1e-7 for scaling factors. Fig2 shows the obtained results. There are two problematic issues. First, notable accuracy loss compared to full precision model. Second, it does not outperform the layer-wise approach shown in Fig3 (almost the same results). A potential reason for both issues is overfitting, i.e., the model is yet too big even after it get ternarized, as a result, we cannot benefit from full capacity of our filter-wise method and distinguish between the filter-wise and layer-wise methods. Furthermore, Fig5 and Fig6 show the top1 accuracy and loss value for the method that is further enriched by incremental technique. For the case of loss value, graphs from top to bottom relate to round one to three, and for the accuracy, the graphs are tied in reverse direction. We set the accumulated portions of ternarized weights at subsequent steps as [0.5, 0.75, 1]. As the incremental approach moves forward, the experimental results indicate decent starting phase at following round that somehow justify the idea that this approach gently traverse between subsequent training rounds by providing good initialization for the next round. However, after quite a few training iteration it fails to achieve satisfying performance by the end of last round. This degradation might be caused by overfitting as discussed earlier. Moreover, I implemented the approach proposed by [18] as it also performs filter-wise ternarization but does not learn the scaling factor. The obtained results show that it works well in this circumstances and does not indicate any overfitting problem.

A potential solution to cope with this situation might be examining and training the ternary model on ImageNet rather than Tiny-ImageNet to avoid overfitting. At the moment, I am working on this possible solution and running the code on Compute Canada in a reasonable amount of time, but not optimistic that the final results will be satisfying and competitive.

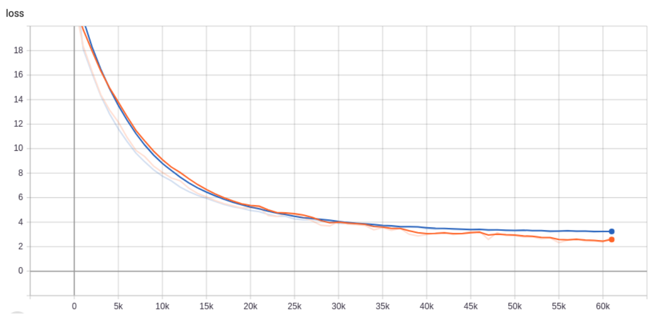
3.2. ImageNet

4. Discussion

5. Conclusion

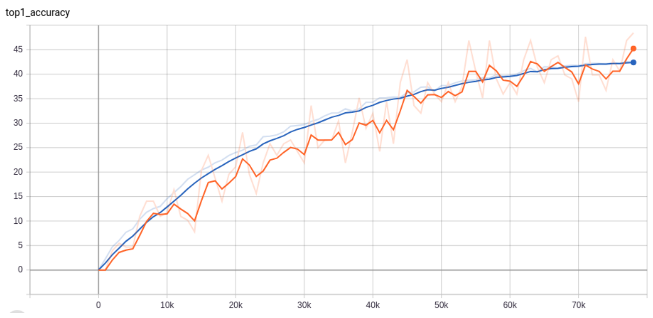


(a) top1 accuracy

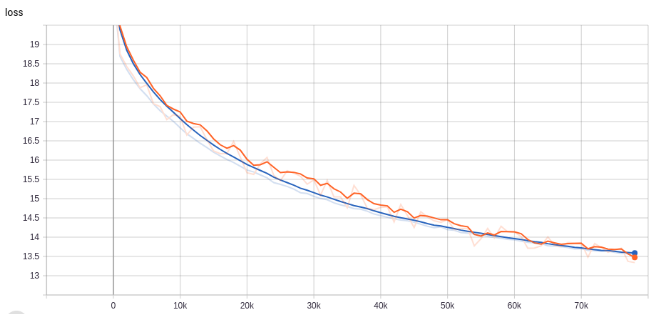


(b) loss value

Figure 1: Full precision ResNet101 trained on tiny-imagenet dataset

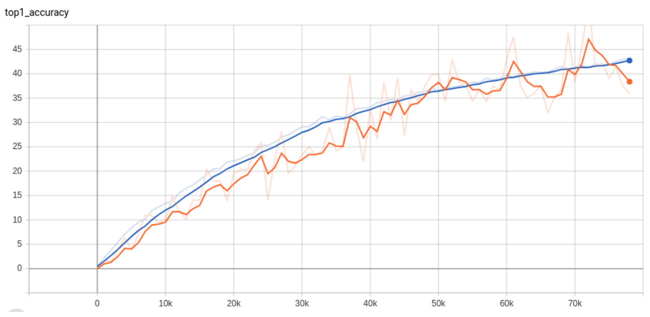


(a) top1 accuracy

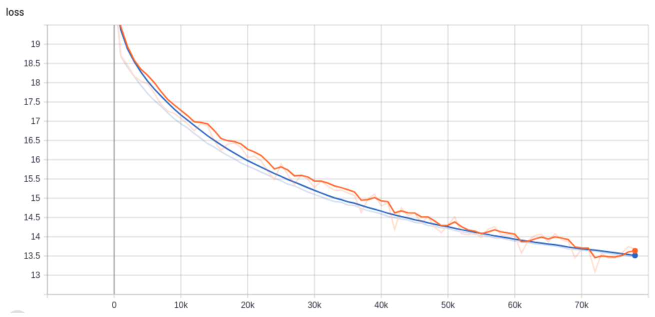


(b) loss value

Figure 2: Filter-wise double-factor trained ternary ResNet101 trained on tiny-imagenet dataset

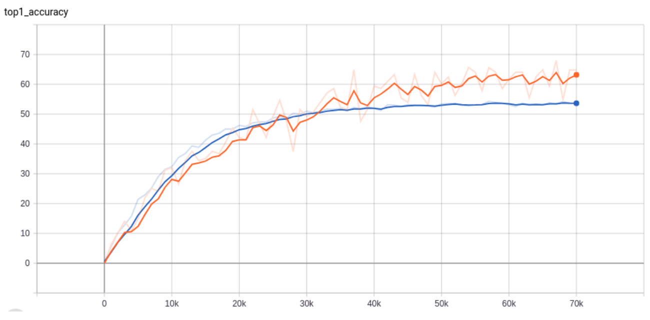


(a) top1 accuracy

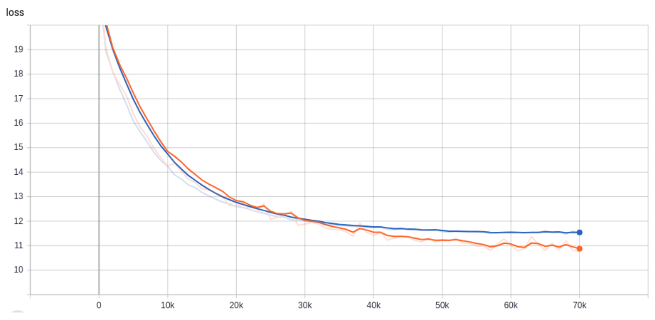


(b) loss value

Figure 3: Layer-wise double-factor trained ternary ResNet101 trained on tiny-imagenet dataset

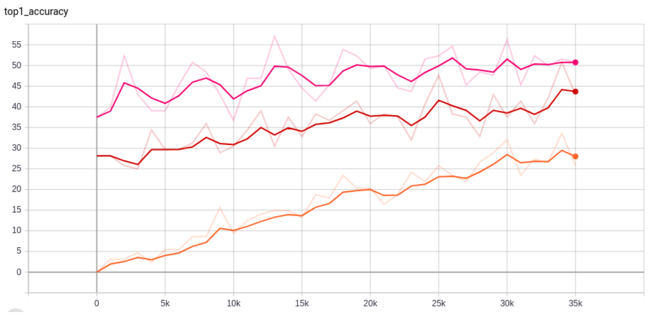


(a) top1 accuracy

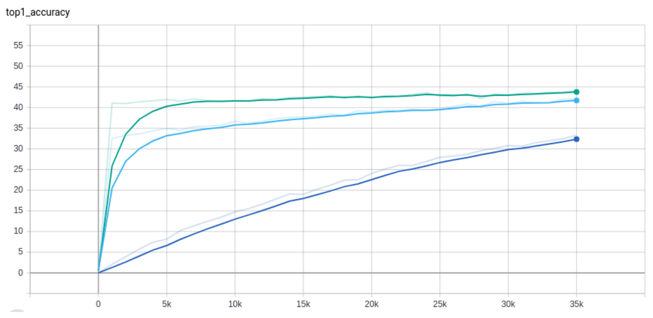


(b) loss value

Figure 4: Filter-wise single-factor ternary ResNet101 trained on tiny-imagenet dataset

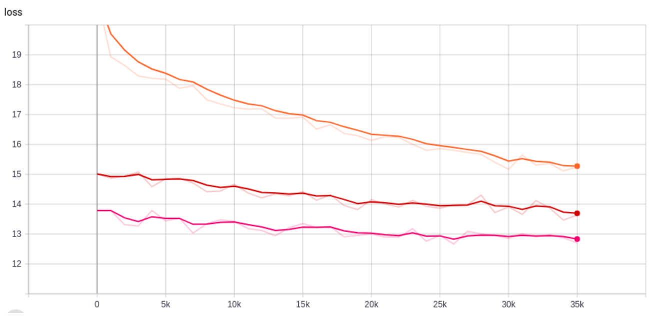


(a) train

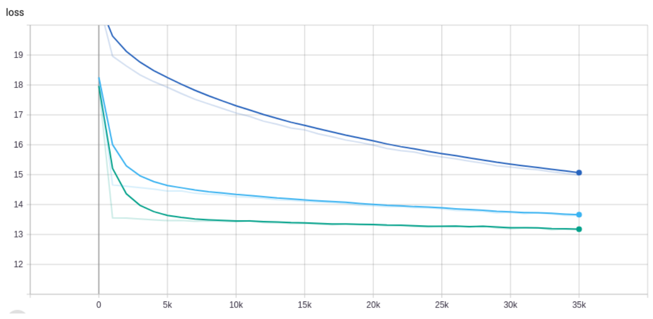


(b) validation

Figure 5: Top1 accuracy of incremental filter-wise double-factor trained ternary ResNet101 on tiny-imagenet dataset



(a) train



(b) validation

Figure 6: Loss value of incremental filter-wise double-factor trained ternary ResNet101 on tiny-imagenet dataset

References

[1] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[3] Y. Guo, A. Yao, H. Zhao, and Y. Chen. Network sketching: Exploiting binary structure in deep cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5955–5963, 2017.

[4] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[7] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[8] L. Hou, Q. Yao, and J. T. Kwok. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600*, 2016.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[10] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2752–2761, 2018.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[12] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[13] J. Kim, S. Park, and N. Kwak. Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems*, pages 2760–2769, 2018.

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

[15] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[16] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[17] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[18] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[20] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[21] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[22] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462*, 2017.

[23] E. Park, J. Ahn, and S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[24] N. Passalis and A. Tefas. Learning deep representations with probabilistic knowledge transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 268–284, 2018.

[25] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[27] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[28] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[29] F. Tung and G. Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7873–7882, 2018.

[30] Y. Wang, C. Xu, C. Xu, and D. Tao. Adversarial learning of portable student networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[32] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[33] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[34] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[35] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid. Towards effective low-bitwidth convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7920–7928, 2018.