SATB-Nets: Training Deep Neural Networks with Segmented Asymmetric Ternary and Binary Weights

Shuai Gao[1](✉), JunMin Wu[2](✉), Da Chen[1], and Jie Ding[2]

[1] School of Software Engineering, University of Science and Technology of China, SuZhou, China
{gshuai16,soloda}@mail.ustc.edu.cn

[2] Department of Computer Science and Technology, University of Science and Technology of China, HeFei, China
jmwu@ustc.edu.cn, dj1993@mail.ustc.edu.cn

Abstract. Deep convolutional neural networks (CNNs) are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources efficiently. To address this limitation, we introduce SATB-Nets, a method which trains CNNs with segmented asymmetric ternary weights for convolutional layers and binary weights for the fully-connected layers. We compare SATB-Nets with previous proposed ternary weight networks (TWNs), binary weight networks (BWNs) and full precision networks (FPWNs) on CIFAR-10 and ImageNet datasets. The result shows that our SATB-Nets model outperforms full precision model VGG16 by 0.65% on CIFAR-10 and achieves up to 29× model compression rate. On ImageNet, there is 31× model compression rate and only 0.15% accuracy degradation over the full-precision AlexNet model of Top-1 accuracy.

Keywords: Deep convolutional neural networks
Segmented asymmetric ternary and binary weights
Model compression · Embedded efficient neural networks

1 Introduction

Deep Neural Networks (DNNs) have inexorably pushed the amazing performances in lots of application domains including but not limited to the speech recognition [1,2] and computer vision, mainly including object recognition [3,4,6,23] and object detection [7,8,10]. A particular type of networks, named Convolution Neural Networks (CNNs), are being deployed to real world applications on smart phones and other embedded devices. However, it is difficult to deploy these computationally intensive and memory-intensive CNNs on embedded devices which are both computational resources limited and storage limited.

1.1 Binary Weight Networks and Model Compression

To address the storage and computational issues [5,21], methods that seek to binarize weights or activations in DNNs models have been proposed. BinaryConnect [11] binarizes the weights to $\{+1, -1\}$ with a single sign function. Binary Weight Networks [12] improve the models' capacity by adding an extra scaling factor on the basis of the previous method. BinaryNet [12] and XNOR-Net [13] binarize not only weights but also activations as extensions of the previous methods. These models eliminate most of the multiplication operations in the forward and backward propagations [16] and model compression rate achieves up to 32×, but there are also considerable accuracy loss.

1.2 Ternary Weight Networks and Model Compression

Nowadays, more and more researchers are engaged in the quantization of 2-bit neural networks especially the ternary weights quantization. Ternary weights networks (TWNs) [14] were introduced with the weights constrained to $\{-1, 0, +1\}$ to maximize scale model compression and minimize the precision loss of the model as far as possible. Compared with the previous binary quantization network, the accuracy loss has been reduced obviously because of the increased weights precision. However, there are also some tricks to improve the capacity of ternary weights networks with the different scaling factors for positive and negative weights.

We optimize the previous methods [14,20] by proposing Segmented Asymmetric Ternary and Binary Weights Networks (SATB-Nets) to explore higher model capacity and model compression rate. For each layer, we segment the weights vector space into many disjoint subspaces. In each subspace, we confine weights to three values $\{+W_{ls}^{pt}, 0, -W_{ls}^{nt}\}$ for convolutional (CONV) layers and two values $\{+W_{ls}^{pb}, -W_{ls}^{nb}\}$ for fully-connected (FC) layers, which can also be encoded with two bits and a single bit. Compared with TWNs [14] and BWNs [11] quantization method, our SATB-Nets are able to explore the local redundancy structure better and gain more stronger expressive abilities leading to better performance. In addition, the fixed scaling factors $\{+W_{ls}^{p*}, 0, -W_{ls}^{n*}\}$ provide more possibilities for computing acceleration.

2 Segmented Asymmetric Ternary and Binary Weights Networks

We will detailedly introduce how to obtain Segmented Asymmetric Ternary and Binary Weights Networks (SATB-Nets) and train them efficiently in this section.

2.1 Segmentation

Product quantization (PQ) [18] partitions the vector space into many disjoint subspaces to explore the redundancy of structures in vector space. Authors of

[9] proposes the segmentation of the weight matrix and then the performance of quantization in each subspace. Similarly, we partition weight matrix into several submatrices to improve the expression ability of the quantized networks:

$$W = [W^1, W^2, ..., W^k], \tag{1}$$

Where $W \in R^{m*n}$ and $W^i \in R^{m*(n/k)}$ assuming n is divisible by k. We can quantify each submatrix W^i with ternary and binary value. More segments lead to higher model capacity but will aggressively increase the codebook size. So, by using the same trick as described in [9], we fixed the number of segments k to 8 to keep a satisfying balance between compression rate and output precision loss of the networks.

2.2 Asymmetric Binary Weights for Fully-Connected(FC) Layers

We constrain the full precision weights W_{lsi} (lth layers, sth segments and ith parameters) to binary weights with values belong to $\{+W_{ls}^{pb}, -W_{ls}^{nb}\}$. The quantization function is shown in (2).

$$w_{lsi}^b = f_b(w_{lsi}^b) = \begin{cases} +W_{ls}^{pb} & w_{lsi} \geq 0 \\ -W_{ls}^{nb} & w_{lsi} < 0 \end{cases} \tag{2}$$

Here 0 is threshold and $\{W_{ls}^{pb}, W_{ls}^{nb}\}$ are the scaling factors. In order to get as well performance as possible, the minimization of Euclidian distance between the floating-point weights W_{ls} and binary weights W_{ls}^b is adopted and the optimization problem is transformed to (3):

$$\begin{cases} W_{ls}^* = arg\min J(W_{ls}^*) = arg\min \left\| W_{ls} - W_{ls}^b \right\|_2^2 \\ s.t. W_{ls}^* > 0; w_{lsi}^b \in \{+W_{ls}^{pb}, 0, -W_{ls}^{nb}\}; i = 1, 2, ..., n; s = 1, 2, ..., k \end{cases} \tag{3}$$

Substitute the binary function (2) into the formula (3), we can get the expression as (4):

$$J(W_{ls}^*) = \left\| W_{ls} - W_{ls}^b \right\|_2^2 = \sum_{i \in I_*}^{n_s} \left| |w_{lsi}| - W_{ls}^* \right|^2 \tag{4}$$

where I_* =$\{I_p, I_n\}$, $I_p = \{i | w_{lsi} \geq 0\}$, $I_n = \{i | w_{lsi} < 0\}$. According to (4), It is not complicated to obtain binary weights from the floating-point weights as (5):

$$W_{ls}^* = \frac{1}{|I_*|} \sum_{i \in I_*}^{n_s} |w_{lsi}| \tag{5}$$

where $|I_*|$ denotes the number of elements in I_* in each segment.

2.3 Asymmetric Ternary Weights for Convolutional (CONV) Layers

Similarly, we also constrain the floating-point weights W_{lsi} (1th layers, sth segments and ith parameters) to ternary weights with values belong to $\{+W_{ls}^{pt}, 0, -W_{ls}^{nt}\}$. The quantization function is shown in (6).

$$w_{lsi}^{t} = f_t(w_{lsi}^{t}|\Delta_{ls}^{p}, \Delta_{ls}^{n}) = \begin{cases} +W_{ls}^{pt} & w_{lsi} > \Delta_{ls}^{p} \\ 0 & -\Delta_{ls}^{n} \le w_{lsi} \le \Delta_{ls}^{p} \\ -W_{ls}^{nt} & w_{lsi} < -\Delta_{ls}^{n} \end{cases} \tag{6}$$

Here $\{\Delta_{ls}^{P}, \Delta_{ls}^{n}\}$ are the threshold and $\{W_{ls}^{pt}, W_{ls}^{nt}\}$ are the scaling factors. The optimization problem is formulated as (7):

$$\begin{cases} W_{ls}^{*} = arg\min J(W_{ls}^{*}) = arg\min \|W_{ls} - W_{ls}^{t}\|_2^2 \\ \\ s.t. W_{ls}^{*} > 0; W_{lsi}^{t} \in \{+W_{ls}^{pt}, 0, -W_{ls}^{nt}\}; i = 1, 2, ..., n; s = 1, 2, ..., k \end{cases} \tag{7}$$

Substitute the ternary function (6) into the formula (7), we can get the expression as (8):

$$J(W_{ls}^{*}) = \left\|W_{ls} - W_{ls}^{t}\right\|_2^2 = \left|I_{\Delta_{ls}^{*}}\right| * (W_{ls}^{*})^2 - 2 * (\sum_{i|i \in I_{\Delta_{ls}^{*}}}^{n_s} |w_{lsi}|) * (W_{ls}^{*}) + C \tag{8}$$

Where $I_{\Delta_{ls}^{*}}$ =$\{I_{\Delta_{ls}^{p}}, I_{\Delta_{ls}^{n}}\}$, $I_{\Delta_{ls}^{p}} = \{i|w_{lsi} > \Delta_{ls}^{p}\}$, $I_{\Delta_{ls}^{n}} = \{i|w_{lsi} < -\Delta_{ls}^{n}\}$ and $|I_{\Delta_{ls}^{*}}|$ denotes the number of elements in $I_{\Delta_{ls}^{*}}$ in each segment. Δ_{ls}^{p} and Δ_{ls}^{n} are independent together. $C = \sum_{i}^{n_s} |w_{lsi}|^2$ is a $\{W_{ls}^{pt}, W_{ls}^{nt}\}$-independent constant. Therefore, our scaling factors $\{W_{ls}^{pt}, W_{ls}^{nt}\}$ can be simplified to:

$$W_{ls}^{*} = arg\min J(W_{ls}^{*}) = arg\min(\left|I_{\Delta_{ls}^{*}}\right| * (W_{ls}^{*})^2 - 2 * (\sum_{i|i \in I_{\Delta_{ls}^{*}}}^{n_s} |w_{lsi}|) * (W_{ls}^{*})) \tag{9}$$

According to (9), It is not complicated to obtain tenary weights from the floating-point weights as (10):

$$W_{ls}^{*} = \frac{1}{\left|I_{\Delta_{ls}^{*}}\right|} \sum_{i|i \in I_{\Delta_{ls}^{*}}}^{n_s} |w_{lsi}| \tag{10}$$

Here $\{\Delta_{ls}^{p}, \Delta_{ls}^{n}\}$ are both positive values. There is no straightforward solutions to figure out Δ_{ls}^{p} and Δ_{ls}^{n} as [17]. But values are generated from uniform or normal distribution empirically, adopting the method mentioned in [14], the thresholds are as following:

$$\Delta_{ls}^{*} \approx 0.7 * \frac{1}{|I^{*}|} \sum_{i|i \in I^{*}}^{n_s} |w_{lsi}| \tag{11}$$

Where $I^* = \{I^p, I^n\}$, $I^p = \{i|w_{lsi} \geq 0|i = 1, 2, ..., n_s\}$, $I^n = \{i|w_{lsi} < 0|i = 1, 2, ..., n_s\}$. Finally, by substituting (10) and (11) to (6), Ternary weights can be easily obtained from the floating-point weights.

2.4 Heterogeneous Quantized Weights Structure

In order to achieve a good balance between compression rate and accuracy, we train CNNs with ternary weights for convolutional layers and binary weights for the fully-connected layers. On the one hand, these densely and highly redundancy fully-connected layers take up most of the parameters, binarization is more helpful in removing redundancy and higher proportion of compression. On the other hand, [21] shows that convolutional layers require more bits of precision than fully-connected layers, so ternary weights for convolutional layers improve the expression capacity. In addition, the quantization values of zero for convolutional layers reduce the calculation of the multiplication to accelerate the networks.

2.5 Train the SATB-Nets with Stochastic Gradient Descent (SGD) Method

Stochastic Gradient Descent (SGD) algorithm is used as the training algorithm for SATB-Nets, about which more detail is shown in Algorithm 1.

The whole training process is almost the same as normal training method, except that segmented asymmetric ternary weights for convolutional (CONV) layers and binary weights for the fully-connected (FC) layers are used in forward propagation (**step 1**) and backward derivation (**step 2**), which is similar to training method as BinaryConnect [11]. In order to overcome the difficulty of convergence of models using quantized weights, we reserved the full precision floating-point weights to update weights to obtain the tiny changes in each iteration (**step 3**).

In addition, Batch Normalization (BN) [24] and learning rate scaling, as two useful tricks, are adopted. We also use momentum for acceleration.

3 Experiments

In this section, we benchmark SATB-Nets with full precision weights networks (FPWNs), binary weights networks (BWNs) and Ternary Weights Networks (TWNs) on the small scale datasets (CIFAR-10) and the large scale dataset (ImageNet datasets). We adopt the VGG [6] networks on Cifar-10 and the AlexNet [3] on ImageNet. To be fair, the following terms are identical: network architecture, learning rate scaling procedure (multi-step), optimization method (SGD with momentum) and regularization method (L2 weight decay). We conjecture that SATB-Nets have sufficient expressiveness in the depth networks and adopt the data augmentation and the sparse weights like dropout [15] to prevent overfitting. In addition, all the neural networks are deployed on framework of Caffe [25]. For more detailed configurations, we can see Table 1.

Algorithm 1. Training a DNNs with SATB-Nets. **L** is the number of layers and **K** is the number of segments. Full precision weights for layer l and segment s are W_{ls} and b_{ls} and the output is a_{ls}. **J** is the cost function of networks and **Binary**(W_{ls}^b) and **Ternary**(W_{ls}^t) mean to quantize weights to binary in fully connected layers (FC) and ternary values in convolution layers (CONV).

```
Begin
  1. Forward propagation:
  for $l \leftarrow 1$ to $L-1$ do
      for $s \leftarrow 1$ to $K-1$ do
          if $CONV$ then
              $W_{ls}^t \leftarrow \textbf{Ternary}(W_{ls})$
              $a_{l+1s} \leftarrow f\left(W_{ls}^t * a_{ls} + b_{ls}\right)$
          end if
          if $FC$ then
              $W_{ls}^b \leftarrow \textbf{Binary}(W_{ls})$
              $a_{l+1s} \leftarrow f\left(W_{ls}^b * a_{ls} + b_{ls}\right)$
          end if
      end for
  end for
  2. Backward derivation:
  for $l \leftarrow 1$ to $L-1$ do
      for $s \leftarrow 1$ to $K-1$ do
          if $CONV$ then
              $\frac{\partial J}{\partial a_{ls}} \leftarrow ((W_{ls}^t)^T * \frac{\partial J}{\partial a_{l+1s}}) \circ f'$
              // $\circ$ means element-wise product
          end if
          if $FC$ then
              $\frac{\partial J}{\partial a_{ls}} \leftarrow ((W_{ls}^b)^T * \frac{\partial J}{\partial a_{l+1s}}) \circ f'$
          end if
          $\frac{\partial J}{\partial W_{ls}} \leftarrow \frac{\partial J}{\partial a_{l+1s}} * (a_{ls})^T$
          $\frac{\partial J}{\partial b_{ls}} \leftarrow \frac{\partial J}{\partial a_{l+1s}}$
      end for
  end for
  3. Weights update:
  for $l \leftarrow 1$ to $L-1$ do
      $W_{ls} \leftarrow W_{ls} - \eta * \frac{\partial J}{\partial W_{ls}}$
      $b_{ls} \leftarrow b_{ls} - \eta * \frac{\partial J}{\partial b_{ls}}$
  end for
End
```

3.1 VGGNets on CIFAR-10

CIFAR-10 is a benchmark image classification dataset which consists of 60K 32×32 color images and Five sixths of them belong to a training set and the rest belong to the test set. To prevent over-fitting while training VGG [6] networks, data-augmentation is used following [4]. A random 32×32 crop is from the padded images on which 4 pixels are padded each side. The cropped images

Table 1. Networks and some hyper-parameters of them on datasets

	Cifar-10			ImageNet
Networks	VGG10	VGG13	VGG16	AlexNet
Weight decay	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}
Momentum	0.9	0.9	0.9	0.9
Mini-batch size of BN	100	100	100	256
Learning rate	0.01	0.01	0.01	0.0001
Learning rate decay epochs (0.1)	50, 100, 150, 200			50, 60

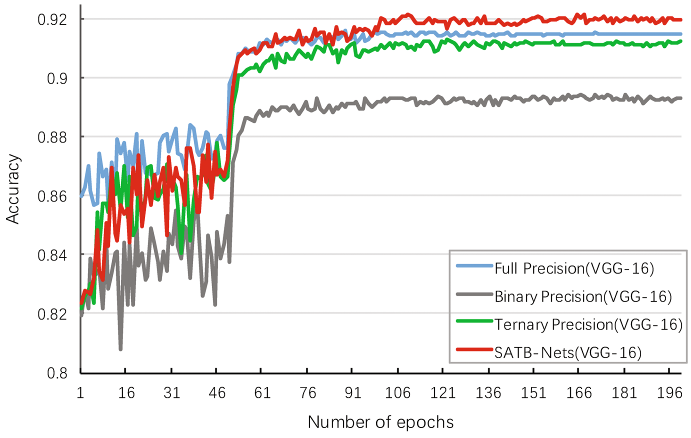

Fig. 1. Validation accuracy curves of VGG16 on Cifar-10

are used for training while original images are for testing. We adopt VGG16 [6] architecture for the experiment firstly. Beside, in order to solve the difficulty of training so deep neural network, we initialize these networks with full-trained full precision model.

We compare SATB-Nets with the FPWNs, BWNs and TWNs. The result (Fig. 1 and Table 2) shows that SATB-Nets from VGG16 outperforms BWNs, TWNs and FPWNs by 2.52%, 1.09% and 0.65% respectively. In the meanwhile, SATB-Nets from VGG10, VGG13 and VGG16 are always outperforming BWNs and TWNs.

To our surprise, the SATB-Nets constrained from VGG13 and VGG16 outperform the full precision weights networks. According to our analysis, we conjecture that our SATB-Nets have adequate capacity for expression and the sparse weights networks prevent over-fitting like dropout [15].

For the more sufficient experimental verification, we expand the experiment to VGG13 removing last 3 convolutional layers of VGG16 [6] and VGG10

Table 2. Validation accuracy of VGGNet on CIFAR-10 (%)

Model	FPWNs	BinaryNet	TWNs	SATB-Nets	Improvements(%)
VGG-10	**90.45**	88.76	89.50	**90.40**	**−0.05**/2.64/0.90
VGG-13	**91.25**	89.09	90.60	**91.96**	**+0.71**/2.87/1.36
VGG-16	**91.50**	89.63	91.06	**92.15**	**+0.65**/2.52/1.09

Table 3. Compression ratio for VGG-16 (Byte)

Layer	Full-weights	BinaryNet	TWNs	SATB-Nets
CONV	58.84 M	1.84 M	3.68 M	3.68 M
FC	494.52 M	15.44 M	30.92 M	15.44 M
Total	553.36 M	17.28 M (32×)	34.24 M (16×)	19.12 M (**29**×)

removing last 6 convolutional layers. The results met our expectations which are listed in Table 2. In the meanwhile, Table 3 shows the compression ratio of VGG-16.

3.2 AlexNet on ImageNet

We further examine the performance of SATB-Nets on the ImageNet ILSVRC-2012 dataset, which has over $1.2M$ training examples and $50K$ validation examples. We use the AlexNet Caffe model [26] as the reference model. Beside, in order to solve the difficulty of training so deep neural network, we initialize these networks with full-trained full precision model.

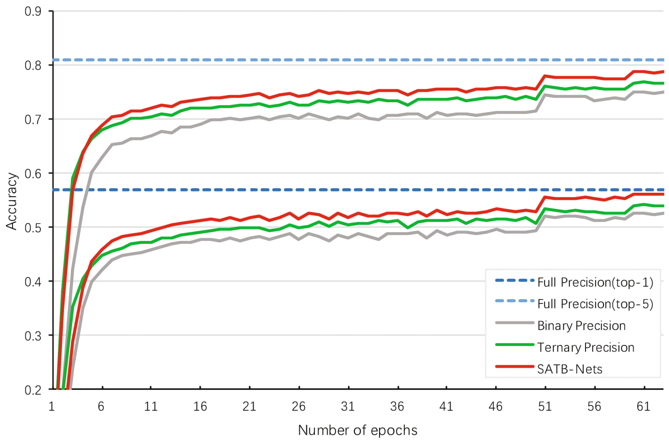

Fig. 2. Validation accuracy curves of AlexNet on ImageNet

Table 4. Validation accuracy of AlexNet on ImageNet(%)

Accuracy	FPWNs	BWNs	TWNs	SATB-Nets
Top-1	**56.72**	52.65	54.01	**56.57**
Top-5	**80.17**	74.89	76.43	**78.76**

Table 5. Compression statistics for AlexNet (Byte)

Layer	Full-weights	BinaryNet	TWNs	SATB-Nets
CONV	9.32 M	0.28 M	0.60 M	0.60 M
FC	234.52 M	7.32 M	14.64 M	7.32 M
Total	243.84 M	7.32 M (32×)	15.24 M (16×)	7.92 M (**31**×)

Our training curves are shown in Fig. 2, the complete result (Fig. 2 and Table 4) shows that SATB-Nets reaches the top-1 validation accuracy of 56.57% which has only 0.15% accuracy degradation over full precision counterpart.

Tables 3 and 5 show the compression ratio of VGG-16 and AlexNet. SATB-Nets achieve up to 29× and 31× model compression rate respectively which are closed to the binary weights compression without impacting accuracy.

4 Conclusion

In this paper, we propose ternary and binary weights networks optimization problems. Next, We propose SATB-Nets which nearly achieve up to binary compression ratio. Meanwhile, experiments show that benchmarks demonstrate the superior performance of the method which we proposed. Next step, we will apply the method to more datasets and models to more deeply explore the relationships between the capacity of networks and the quantized values.

Acknowledgment. The National Key Research and Development Program of China (Grants No. 2016YFB1000403).

References

1. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N.: Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
2. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
3. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the Annual Conference on Neural Information Processing Systems, pp. 1097–1105 (2012)

4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778. IEEE Computer Society (2016)
5. Esser, S.K., Merolla, P.A., Arthur, J.V., Cassidy, A.S., Appuswamy, R., Andreopoulos, A.: Convolutional networks for fast, energy-efficient neuromorphic computing. In: Proceedings of the National Academy of Sciences of the USA, pp. 1441–1446 (2016)
6. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint arXiv:1409.1556
7. Girshick, R., Donahue, J., Darrell, T., et al.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
8. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: International Conference on Neural Information Processing Systems. vol. 39, pp. 91–99. MIT Press, Cambridge (2015)
9. Gong, Y., Liu, L., Yang, M., et al.: Compressing deep convolutional networks using vector quantization (2014). arXiv preprint arXiv:1412.6115
10. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1440–1448 (2015)
11. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Proceedings of the Annual Conference on Neural Information Processing Systems, pp. 3123–3131 (2015)
12. Hubara, I., Courbariaux, M., Soudry, D., et al.: Binarized neural networks. In: Proceedings of the Annual Conference on Neural Information Processing Systems, pp. 4107–4115 (2016)
13. Rastegari, M., Ordonez, V., Farhadi, A., et al.: XNOR-Net: Imagenet classification using binary convolutional neural networks. In: Proceedings of the European Conference on Computer Vision, pp. 525–542 (2016)
14. Li, F., Zhang, B., Liu, B.: Ternary weight networks (2016). arXiv preprint arXiv:1605.04711
15. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
16. Lin, Z., Courbariaux, M., Memisevic, R., Bengio, Y.: Neural networks with few multiplications (2015). arXiv preprint arXiv:1510.03009
17. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights +1, 0, and −1 (2014). arXiv preprint arXiv:1405.3866
18. Jgou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**(1), 117–128 (2010)
19. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: In Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, pp. 562–570 (2015)
20. Ding, J., Wu, J.M., Wu, H.: Asymmetric ternary networks. In: International Conference on TOOLS with Artificial Intelligence IEEE Computer Society, pp. 61–65 (2017)
21. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (2015). arXiv preprint arXiv:1510.00149
22. Deng, J., Dong, W., Socher, R., et al.: Imagenet: a large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009)

23. Szegedy, C., Liu, W., Jia, Y., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
24. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine Learning, pp. 448–456 (2015)
25. Jia, Y., Shelhamer, E., Donahue, J., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the International Conference on Multimedia Retrieval, pp. 675–678 (2014)
26. BVLC: Caffe model zoo. http://caffe.berkeleyvision.org/model_zoo