Adaptive Binary-Ternary Quantization

Ryan Razani, Grégoire Morin, Eyyüb Sari, and Vahid Partovi Nia
Huawei Noah's Ark Lab
ryan.razani@huawei.com

Abstract

Neural network models are resource hungry. It is difficult to deploy such deep networks on devices with limited resources, like smart wearables, cellphones, drones, and autonomous vehicles. Low bit quantization such as binary and ternary quantization is a common approach to alleviate this resource requirements. Ternary quantization provides a more flexible model and outperforms binary quantization in terms of accuracy, however doubles the memory footprint and increases the computational cost. Contrary to these approaches, mixed quantized models allow a trade-off between accuracy and memory footprint. In such models, quantization depth is often chosen manually, or is tuned using a separate optimization routine. The latter requires training a quantized network multiple times. Here, we propose an adaptive combination of binary and ternary quantization, namely Smart Quantization (SQ), in which the quantization depth is modified directly via a regularization function, so that the model is trained only once. Our experimental results show that the proposed method adapts quantization depth successfully while keeping the model accuracy high on MNIST and CIFAR10 benchmarks.

1. Introduction

Deep Neural Network (DNN) models have achieved tremendous attraction due to their success on a wide variety of tasks, including computer vision, automatic speech recognition, natural language processing, and reinforcement learning [3]. More specifically, in computer vision DNN have led to a series of breakthrough for image classification [6], [16], [17], and object detection [14], [11], [15]. DNN models are computationally intensive and require large memory to store the model parameters. Computation and storage resource requirement becomes the main obstacle to deploy such models in many edge devices due to lack of memory, computation power, energy, etc. This motivated the researchers to develop compression techniques to reduce the cost for such models.

Recently, several techniques have been introduced in the literature to solve the storage and computational limitations of the edge devices. Among them, quantization methods focus on representing the weights of a neural network in lower precision than the usual 32-bits float representation, saving on the memory footprint of the model. Binary quantization [1], [4], [13], [19], [9], [2] represent weights with 1 bit precision and ternary quantization [10], [8], [20] with 2 bits precision. While the latter frameworks lead to significant memory reduction compared to their full precision counterpart, they are constrained to quantize the model with 1 bit or 2 bits, on demand. We relax this constraint, and present Smart Quantization (SQ) that allows adapting layers to 1 bit and 2 bits while training the network. Consequently, this approach automatically quantizes weights into binary or ternary depending upon a trainable control parameter. We show that this approach leads to mixed bit precision models that beats ternary networks both in terms of accuracy and memory consumption. Here we only focus on quantizing layers because it is more feasible to implement layer-wise quantization at inference time after training. However, this method can be also adapted for mixed precision training of sub-network, block, filter, or weight.

2. Related Work

There are two main components in DNN models, namely, weights and activations. These two components are usually computed in full precision, i.e. floating point 32-bits. This work focuses on quantizing the weights of the network, i.e. generalizing BinaryConnect (BC) [1] and Ternary Weight Network (TWN) [8] toward automatic 1 or 2 bits mixed-precision using a single training algorithm.

In BC [1], the real value weights, w, are binarized to $w^b \in \{-1, +1\}$ during the forward pass. To map a full precision weight to a binary weight, the deterministic $sign$ function is used,

$$w^b = \text{sign}(w) = \begin{cases} +1 & w \geq 0, \\ -1 & w < 0. \end{cases} \tag{1}$$

The derivative of the $sign$ function is zero on $\mathbb{R} \setminus \{0\}$. The sign function has zero gradient which freezes weight up-

dates during back-propagation. To bypass this problem, BC [1] uses a clipped straight-through estimator

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial w^b} \mathbf{1}_{|w|\leq 1}(w) \tag{2}$$

where $\mathcal{L}$ is the loss function and $\mathbf{1}_A(.)$ is the indicator function on the set A. In other words (2) approximates the sign function by the linear function $f(x) = x$ within $[-1, +1]$ and a constant elsewhere. During back propagation, the weights are updated only within $[-1, +1]$. The binarized weights are updated with their corresponding full precision gradients. XNOR-Net [13] adds a scaling factor to reduce the gap between binary and full-precision model's accuracy, defining Binary Weight Network (BWN). The real value weights $\mathbf{W}$ in each layer are quantized as $\mu \times \{-1, +1\}$ where $\mu = \mathbb{E}\big[|\mathbf{W}|\big] \in \mathbb{R}$. DoReFa-Net [19] generalizes the latter work and approximates the full precision weights with more than one bit, while ABCNet [9] approximates weights with a linear combination of multiple binary weight bases.

2.1. Ternary weight networks

Ternary Weight Network (TWN) [8] is a neural network with weights constrained to $\{-1, 0, +1\}$. The weight resolution is reduced from 32 bits to 2 bits, replacing full precision weights with ternary weights. TWN aims to fill the gap between full precision and binary precision weight. Compared to binary weight networks, ternary weight networks are more expressive. In 3×3 weight filter in a convolutional neural network, there are $2^{3\times3} = 512$ possible variations with binary precision and $3^{3\times3} = 19683$ with ternary precision.

TWN [8] finds the closest ternary weights matrix $\mathbf{W}^t$ to its corresponding real value weight matrix $\mathbf{W}$ using

$$\begin{cases} \hat{\mu}, \hat{\mathbf{W}}^t = \underset{\mu, \mathbf{W}^t}{\arg\min} \|\mathbf{W} - \mu \mathbf{W}^t\|_2^2, \\ s.t. \ \ \mu \geq 0, \ \ w_{ij}^t \in \{-1, 0, 1\}, \ \ i, j = 1, 2, ..., n. \end{cases} \tag{3}$$

The ternary weight $\mathbf{W}^t$ is achieved by applying a symmetric threshold Δ

$$\mathbf{W}^t = \begin{cases} +1 & w_{ij} > \Delta, \\ 0 & |w_{ij}| \leq \Delta, \\ -1 & w_{ij} < -\Delta. \end{cases} \tag{4}$$

One may adopt a weight-dependant threshold Δ and a scaling factor μ that approximately solves (3). Similar to BC and BWN schemes; ternary-value weights are only used for the forward pass and back propagation, but not for the parameter updates. At inference, the scaling factor can be folded with the input $\mathbf{X}$ as,

$$\mathbf{X} \odot \mathbf{W} \approx \mathbf{X} \odot (\mu \mathbf{W}^t) = (\mu \mathbf{X}) \odot \mathbf{W}^t, \tag{5}$$

where $\odot$ denotes the convolution.

Trained Ternary Quantization (TTQ) [20] proposes a more general ternary method which reduces the precision of weights in neural network to ternary values. However, TTQ quantizes the weights to asymmetric values $\{-\mu_1, 0, +\mu_2\}$ using two full-precision scaling coefficients μ_1 and μ_2 for each layer of neural network. Consequently, the method achieves better accuracy as opposed to TWN.

Note that μ is regarded as a scaling factor when the network is quantized, during training, within a certain structure (e.g., per layer, per filter, per channel). Varying μ per weight may end up canceling the ternary simplification which requires the same hardware as a full precision network. However, restricting μ in the form of 2^n simplifies multiplication to shift operation to the right or left depending on the sign of the integer value $n \in \mathbb{Z}$, see [18].

Our method provides a compromise between BC and TWN and trains weights with a single trainable scaling factor μ. Weights shift between ternary $\{-\mu, 0, +\mu\}$ and binary $\{-\mu, +\mu\}$. This provides a single algorithm for 1 or 2 bits mixed precision.

2.2. Regularization

Regularization technique is essential to prevent overfitting problem and to obtain robust generalization for unseen data. Standard regularization functions, such as L_2 or L_1 encourage weights to be concentrated about the origin. However, in case of binary network it is more appropriate to have a regularization function to encourage the weights about $\mu \times \{-1, +1\}$, with a scaling factor $\mu > 0$ [12] such as ,

$$R_1(w, \mu) = \big||w| - \mu\big|, \tag{6}$$

A straightforward generalization for ternary quantization can be expressed as,

$$R_2(w, \mu) = \Big|\big||w| - \frac{\mu}{2}\big| - \frac{\mu}{2}\Big|. \tag{7}$$

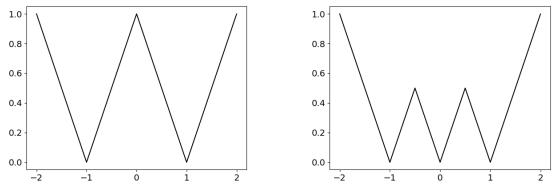
Regularizer (6) encourages weights about $\{-\mu, +\mu\}$, and (7) about $\{-\mu, 0, +\mu\}$. The two functions are depicted in Figure 1. These regularization functions are only useful when the quantization depth is set before training starts. We propose a more flexible version to smoothly move between these two functions using a shape parameter β.

3. Adaptive Quantization

Here we propose a generalized adaptive regularization function that switches between binary regularization of (6) and ternary regularization of (7)

$$\min\Big(\big||w| + \mu\big|^p, \big||w| - \mu\big|^p, \ \tan(\beta)|w|^p\Big), \tag{8}$$

in which μ is a trainable scaling factor, p is the order coefficient denoting the type of regularization function, and



(a) $R_1(w, \mu = 1)$ (b) $R_2(w, \mu = 1)$

Figure 1: Binary and ternary regularizers; R_1 encourages binary weights, with minimums at $\{-\mu, +\mu\}$, and R_2 encourages ternary weights, with minimums at $\{-\mu, 0, +\mu\}$.

$\beta \in (\frac{\pi}{4}, \frac{\pi}{2})$ controls the transition between (6) and (7). As a special case $\beta \to \frac{\pi}{2}$ converges to the binary regularizer (6) and $\beta \to \frac{\pi}{4}$ coincide with the ternary regularizer (7), depicted in Figure 2. A large value of $\tan(\beta)$ repels estimated weights away from zero thus yielding binary quantization, and a small value of $\tan(\beta)$ encourages zero weights. The shape parameter β controls the quantization depth. Quantization depth changes per layer; therefore we let β vary per layer. We recommend to regularize β about $\frac{\pi}{2}$ i.e. preferring binary quantization apriori by adding $|\cot(\beta)|$ to the objective function.

For a single filter $\mathbf{W}$ the regularization function can be expressed by a sum over its elements on row i and column j as,

$$R(\mathbf{W}, \mu, \beta) = \sum_{i=1}^{I} \sum_{j=1}^{J} \min \Big(\big||w_{ij}| + \mu\big|^p, \big||w_{ij}| - \mu\big|^p, \tan(\beta)|w_{ij}|^p \Big) + \gamma |\cot(\beta)|, \tag{9}$$

where γ controls the proportion of binary to ternary layers, i.e., large values of γ contributes to encouraging the layers to form binary values. In each layer, weights are pushed to binary or ternary values, depending on the trained value of the corresponding β. Here we only focus on such regularizer that is constructed using $p = 1$, as the accuracy does not change significantly by varying the value of p.

The introduced regularization function is added to the empirical loss function $L(.)$. The objective function is optimized on set of parameter weights $\boldsymbol{\mathcal{W}}$, set of scaling factors $\boldsymbol{\mu}$, and set of shape parameters $\boldsymbol{\beta}$ using back propagation

$$\mathcal{L}(\boldsymbol{\mathcal{W}}, \boldsymbol{\mu}, \boldsymbol{\beta}) = L(\boldsymbol{\mathcal{W}}) + \sum_{l=1}^{L} \lambda_l \sum_{k=1}^{K} R(\mathbf{W}_{kl}, \mu_{kl}, \beta_l), \tag{10}$$

where k indexes the channel in a convolution network, and l indexes the layer. One may use a different regularization constant λ_l for each layer to keep the impact of the regularization term balanced across layers, indeed different layers may involve different number of parameters. We set $\lambda_l = \frac{\lambda}{\#\mathbf{W}_l}$ where λ is a constant, and $\#\mathbf{W}_l$ is the number of weights in layer l.

We propose to use the same threshold-based function of TWN [8] (4), but with a fixed threshold Δ_l per layer l. Note that [8] proposes a weight-dependant threshold. We enforce weights to only accumulate about $\{-\mu, +\mu\}$ for large β_l. One may set Δ_l to have the same balanced weights in $\{-\mu, 0, +\mu\}$ at initialization for all layers and let the weights evolve during training. Formally, if σ_l is the standard deviation of the initial Gaussian weights in layer l, we propose $\Delta_l = 0.2 \times \sigma_l$. The probability that a single weight lies in the range $[-\Delta_l, \Delta_l]$ is ≈ 0.16. All the weights falling in this range will be quantized as zeros after applying the threshold function.

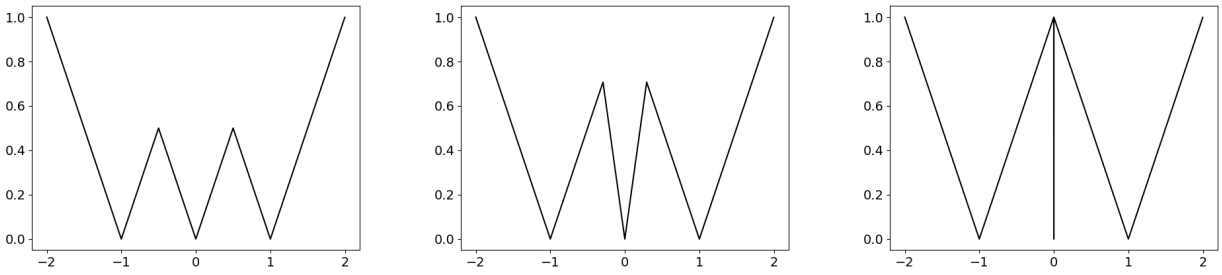
Weights are naturally pushed to binary or ternary values depending on β_l during training. Eventually, a threshold δ close to $\frac{\pi}{2} \approx 1.57$ defines the final quantization depth for each layer.

$$\text{Final quantization depth of layer } l: \begin{cases} \text{Binary} & \beta_l \geq \delta, \\ \text{Ternary} & \beta_l < \delta \end{cases}$$

4. Experiments

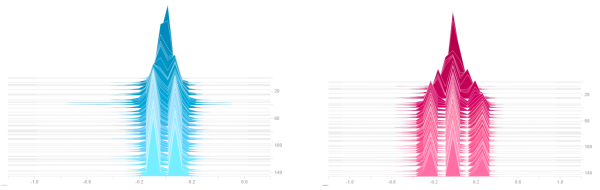
We run experiments on two common image classification tasks MNIST [7] and CIFAR10 [5] datasets. We compare our method, Smart Quantization (SQ), with BinaryConnect (BC) of [1], Binary Weight Networks (BWN) of [13], Ternary Weights Network (TWN) of [8] and also with a Full Precision network (FP). The quality of the compression is measured only in terms of memory, it is difficult to compare mixed precision models, with binary and ternary, in terms of consumed energy as their fair comparison requires specific hardware design. Let n_l be the quantization depth for the layer l and $\#\mathbf{W}_l$ the number of weights in layer l, therefore the compression ratio is derived as, $\frac{\sum_{l=1}^{L} \#\mathbf{W}_l \times 32}{\sum_{l=1}^{L} \#\mathbf{W}_l \times n_l}$. The compression ratio for a binary network is 32, for a ternary network 16, and our approach falls in between.

Our SQ network generalizes binary and ternary regularization in a single regularization function. We present how to control the proportion of binary and ternary layers using γ in (8). Figure 3 clarifies the effect of γ on the weight distribution. When γ is large, β is encouraged towards $\frac{\pi}{2}$ which corresponds to binary quantization. Thus, weights are pushed about $\{-\mu, +\mu\}$ and 0 is removed from the trained values, see Figure 3a. On the contrary, when γ is small, β tends to $\frac{\pi}{4}$ and the weights started including 0 in their values, see Figure 3b. In our experiment, the parameters $\boldsymbol{\mathcal{W}}$, $\boldsymbol{\mu}$ and $\boldsymbol{\beta}$ are trainable; p, δ_l, Δ_l, λ_l, and γ are tuning parameters.



(a) $R(w, \beta = \frac{\pi}{4})$ (b) $R(w, \beta = \frac{3\pi}{8})$ (c) $R(w, \beta = 1.5707)$

Figure 2: Adaptive regularization function. When $\beta \to \frac{\pi}{2}$ the regularization function switches from ternary to binary.



(a) $\gamma = 1 \times 10^{-1}, \beta = 1.57$ (b) $\gamma = 1 \times 10^{-5}, \beta = 0.79$

Figure 3: Effect of γ on the weights distribution of a layer while training. a) binray and b) ternary weight distribution, respectively.

4.1. MNIST

MNIST is an image classification benchmark dataset with 28×28 gray-scale images representing digits ranging from 0 to 9. The dataset is split into 60k training images and 10k testing images. We used the LeNet-5 [7] architecture consisting of 5 layers, 2 convolution followed by maxpooling, stacked with two fully connected layers and a softmax layer at the end. We train the network for 60 epochs using Adam optimizer. We used the initial learning rate of 0.01, but divided by 10 in epoch 15 and 30 to stabilize training. The batch size is set to 64 with L_2 weight decay constant 10^{-4} only for BC, BWC and TWN. The full precision LeNet-5 is trained with no regularization as it provided a superior accuracy. SQ is trained with $\lambda = 10^{-1}$ and $\gamma = 10^{-2}$ and the effective regularization constant is divided by the number of weights in each layer to compensate for the layer size. Validation accuracy for each method is reported in Table 1, as well as the quantization depth, and the overall compression ratio. We observe that SQ network quantized the first two convolutional layers in 1 bit, and the last fully-connected layers in 2 bits. The accuracy improvement and the compression ratio is marginal for simple task and simple architectures. The effect of smart training becomes more visible for more complex tasks with more layers.

Method	Quantization depth per layer (-bits)	Compression ratio	Accuracy (top-1)
BC	1-1-1-1-1	32	99.35
BWN	1-1-1-1-1	32	99.32
TWN	2-2-2-2-2	16	99.38
SQ	1-1-2-2-2	16.3	99.37
FP		1	99.44

Table 1: Smart Quantization (SQ) compared with Binary Connect (BC), Binary Weight Network (BWN), Terneary Weight Network (TWN), and Full Precision (FP) on MNIST dataset.

4.2. CIFAR10

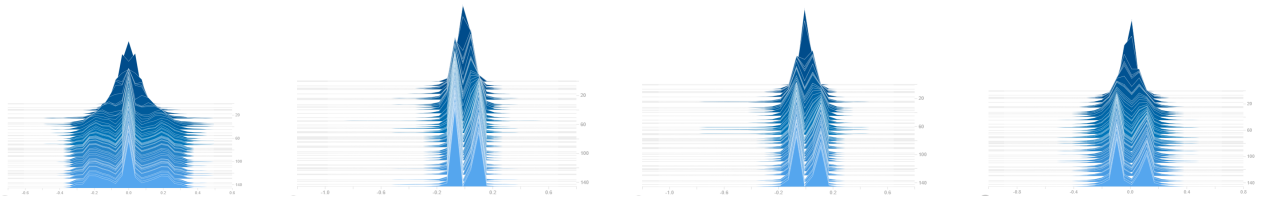
CIFAR10 is an image classification benchmark that contains 32×32 RGB images from ten classes. The dataset is split into 50k training images and 10k testing images. All images are normalized using mean $= (0.4914, 0.4822, 0.4465)$ and std $= (0.247, 0.243, 0.261)$. For the training session, we pad the sides of the images with 4 pixels, then sample a crop of size 32×32, and flip horizontally at random as our augmentation process.

We use two VGG-like architectures, i) VGG-7 architecture defined in [8] in which we apply batch normalization after each layer and use ReLU activations, ii) a standard VGG-16 architecture. We did not quantize the first and the last layers in VGG-16 as the accuracy dropped significantly for all methods.

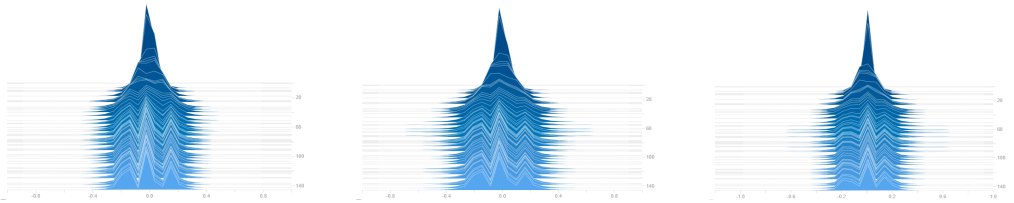
We train the network for 150 epochs, using Adam optimizer with the initial learning rate 0.001 divided by 10 at epochs 40 and 80. The batch size is set to 64 with L_2 weight decay constant 10^{-4}, moreover $\lambda = 0.1$, $\gamma = 10^{-3}$ for SQ. Validation accuracy for each method is reported in Table 2. SQ beats pure 2 bits network TWN, even in terms of accuracy. It recommends three 1 bit layers for VGG-7 and seven 1 bit layers for VGG-16. The compression ratio

Architecture	Method	Quantization depth per layer (-bits)	Compression ratio	Accuracy (top-1)
VGG-7	BC	1-1-1-1-1-1-1	32	92.49
	BWN	1-1-1-1-1-1-1	32	92.42
	TWN	2-2-2-2-2-2-2	16	92.74
	SQ	2-1-1-1-2-2-2	18.3	**92.94**
	FP		1	93.72
VGG-16	BC	32-1-1-1-1-1-1-1-1-1-1-1-1-1-32	31.5	91.92
	BWN	32-1-1-1-1-1-1-1-1-1-1-1-1-1-32	31.5	91.85
	TWN	32-2-2-2-2-2-2-2-2-2-2-2-2-2-32	15.9	92.14
	SQ	32-2-1-1-2-2-2-1-1-1-1-1-2-32	25.1	**92.38**
	FP		1	92.53

Table 2: Smart Quantization (SQ) compared with Binary Connect (BC), Binary Weight Network (BWN), Ternary Weight Network (TWN), and Full Precision (FP) on CIFAR10 dataset.



(a) Layer 1, $\beta_1 = 1.56$ (b) Layer 2, $\beta_2 = 1.57$ (c) Layer 3, $\beta_3 = 1.57$ (d) Layer 4, $\beta_4 = 1.57$



(e) Layer 5, $\beta_5 = 1.00$ (f) Layer 6, $\beta_6 = 0.86$ (g) Layer 7, $\beta_7 = 0.98$

Figure 4: Layer-wise weights distribution in VGG-7 for SQ. The weights are pushed to binary when the shape parameter β is close to $\frac{\pi}{2} \approx 1.57$.

is significantly higher than a ternary network. The weight distribution of each layers are depicted in Figure 4 for the VGG-7 architecture. Weights are pushed to $\{-\mu, +\mu\}$ or $\{-\mu, 0, +\mu\}$ depending on the shape parameter β.

5. Conclusions

We proposed Smart Quantization (SQ), a training method to build a 1 and 2 bits mixed quantized DNN. Depth optimization requires training network multiple times which is costly, especially if the network is complex. However, our proposed method successfully combines quantization with different depths, while training the network only once. We focused on layer-wise quantization, since it is more suitable for mixed-precision inference implementation. However, subnetwork, block, filter, or weight mixed quantization is feasible using a similar algorithm.

SQ makes manual tuning of quantization depth unnecessary. It allows to improve the memory consumption, by automatically quantizing some layers with smaller precision. In some cases, this method even outperforms pure ternary networks in terms of accuracy due to a formal regularization function that shapes trained weights towards mixed-precision. It is well-known that some layers are more resilient to aggressive quantization. Our proposed methodology offers a network similar to pure ternary but gives an insight about which layers can be simplified further by adapting to binary quantization. Scaling the method for complex tasks such as ImageNet and with deeper architectures is challenging and requires further research. A similar methodology can be applied to quantize neural network models designed for other tasks related to speech and text.

References

[1] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems (NIPS)*, pages 3123–3131, 2015. 1, 2, 3

[2] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Regularized binary network training, 2018. 1

[3] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. 1

[4] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems (NIPS)*, pages 4107–4115, 2016. 1

[5] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 3

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017. 1

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. 3, 4

[8] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, 1605.04711, 2016. 1, 2, 3, 4

[9] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in neural information processing systems (NIPS)*, pages 344–352, 2017. 1, 2

[10] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *CoRR*, 1510.03009, 2015. 1

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, 1512.02325, 2015. 1

[12] Vahid Partovi Nia and Mouloud Belbahri. Binary quantizer. *Journal of Computational Vision and Imaging Systems*, 4(1):3–3, 2018. 2

[13] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, 1603.05279, 2016. 1, 2, 3

[14] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, 1506.02640, 2015. 1

[15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, 1506.01497, 2015. 1

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 1409.1556, 2014. 1

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1

[18] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9127–9135, 2018. 2

[19] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, 1606.06160, 2016. 1, 2

[20] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, 1612.01064, 2016. 1, 2