RANDOM PARTITION RELAXATION FOR TRAINING BINARY AND TERNARY WEIGHT NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present Random Partition Relaxation (RPR), a method for strong quantization of the parameters of convolutional neural networks to binary (+1/–1) and ternary (+1/0/–1) values. Starting from a pretrained model, we first quantize the weights and then relax random partitions of them to their continuous values for retraining before quantizing them again and switching to another weight partition for further adaptation. We empirically evaluate the performance of RPR with ResNet-18, ResNet-50 and GoogLeNet on the ImageNet classification task for binary and ternary weight networks. We show accuracies beyond the state-of-the-art for binary and ternary weight GoogLeNet and competitive performance for ResNet-18 and ResNet-50 using a SGD-based training method that can easily be integrated into existing frameworks.

1 INTRODUCTION

Deep neural networks (DNNs) have become the preferred approach for many computer vision, audio analysis and general signal processing tasks. However, they are also known for their associated high computation workload and large model size. These are great hurdles to their wide-spread adoption due to the consequential cost, which is often prohibitive for low-power, mobile and always-on applications.

This concern has driven a lot of research into various DNN topologies and their basic building blocks in order to reduce the required compute cost at a small accuracy penalty. Furthermore, efforts have been made towards compressing the models from often hundreds of megabytes to a size that is suitable for over-the-air updates and does not negatively impact the user experience by taking up lots of storage on consumer devices and long loading times.

Recent research into specialized hardware accelerators has shown that improvements by 10–100$\times$ in energy efficiency over optimized software are achievable (Sze et al., 2017). These accelerators can be integrated into a system-on-chip like those used in smartphones and highly integrated devices for the internet-of-things market. These devices still spend most energy on I/O for streaming data in and out of the hardware unit repeatedly as only a limited number of weights can be stored in working memory—and if the weights fit on chip, local memories and the costly multiplications start dominating the energy cost. This allows devices such as (Andri et al., 2018) achieve an energy efficiency of 60 TOp/s/W for BWN inference even in the mature 65 nm technology. For comparison, Google's Edge TPU achieves 2 TOp/s/W for 8 bit operations.

Quantizing neural networks is crucial to allow more weights to be stored in on-chip working memory or to be loaded more efficiently from external memory, thereby reducing the number of repeated memory accesses to load and store partial results. Complex network compression schemes cannot be applied at this point as decompression is often a lengthy process requiring a lot of energy by itself. Furthermore, by strongly quantizing the network's parameters, the multiplications in the convolution and linear layers can be simplified, replaced with lightweight bit-shift operations, or even completely eliminated in case of binary and ternary weight networks (BWNs, TWNs) (Zhou et al., 2017).

2 RELATED WORK

Extreme network quantization has started with BinaryConnect (Courbariaux et al., 2015) proposing deterministic or stochastic rounding during the forward pass and updating the underlying continuous-valued parameters based on the so-obtained gradients which would naturally be zero almost everywhere.

Then, XNOR-net (Rastegari et al., 2016) successfully trained both binary neural networks (BNNs), where the weight and the activations are binarized, as well as BWNs, with a clear jump in accuracy over BinaryConnect by means of dynamic (input-dependent) normalization and for the first time reporting results for a deeper and more modern ResNet topology.

Shortly after, (Li et al., 2016) presented *ternary weight networks* (TWNs), where they introduced learning the quantization thresholds while keeping the quantization levels fixed and showing a massive improvement over previous work and a top-1 accuracy drop of only 3.6% on ImageNet, making TWNs a viable approach for practical inference.

Thereafter, (Zhu et al., 2017) introduced *trained ternary quantization* (TTQ), relaxing the constraint of the weights being scaled values of $\{-1, 0, 1\}$ to $\{\alpha_1, 0, \alpha_2\}$.

A method called *incremental network quantization* (INQ) was developed in (Zhou et al., 2017), making clear improvements by neither working with inaccurate gradients or stochastic forward passes. Instead, the network parameters were quantized step-by-step, allowing the remaining parameters to adapt to the already quantized weights. This further improved the accuracy for TWNs and fully matched the accuracy of the baseline networks with 5 bit and above.

Last year, (Leng et al., 2018) presented a different approach to training quantized neural networks by relying on the *alternating direction method of multipliers* (ADMM) more commonly used in chemical process engineering. They reformulated the optimization problem for quantized neural networks with the object function being a sum of two separable objectives and a linear constraint. ADMM alternatingly optimizes each of these objectives and their dual to enforce the linear constraint. In the context of quantized DNNs, the separable objectives are the optimization of the loss function and the enforcement of the quantization constraint, which results in projecting the continuous values to their closest quantization levels. While ADMM achieves state-of-the-art results to this day, it requires optimization using the extragradient method, thereby becoming incompatible with standard DNN toolkits and hindering widespread adoption.

A few months ago, *quantization networks* (QNs) was introduced in (Yang et al., 2019). They pursue a very different approach, annealing a smoothed multi-step function the hard steps quantization function while using L2-norm gradient clipping to handle numerical instabilities during training. They follow the approach of TTQ and learn the values of the quantization levels.

3 RPR: RANDOM PARTITION RELAXATION TRAINING

In this section, we describe the intuition behind RPR, its key components and their implementation.

When training DNNs, we optimize the network's parameters $\boldsymbol{w} \in \mathbb{R}^d$ to minimize a non-convex function f,

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} f(\boldsymbol{w}). \tag{1}$$

This has been widely and successfully approached with stochastic gradient descent-based methods for DNNs in the hope of finding a good local optimum close to the global one of this non-convex function.

As we further constrain this optimization problem by restricting a subset of the parameters to take value in a finite set of quantization levels $\mathbb{L}$, we end up with a mixed-integer non-linear program (MINLP):

$$\min_{\boldsymbol{w}_q, \boldsymbol{w}_c} f(\boldsymbol{w}_q, \boldsymbol{w}_c) \qquad \text{s.t.} \quad \boldsymbol{w}_q \in \mathbb{L}^{d_q}, \quad \boldsymbol{w}_c \in \mathbb{R}^{d_c}, \tag{2}$$

where $\boldsymbol{w}_q$ are the quantized (e.g., filter weights) and $\boldsymbol{w}_c$ the continuous parameters (e.g., biases, batch norm factors) of the network. Common sets of quantization levels $\mathbb{L}$ are symmetric uniform

with or without zero ($\{0\} \cup \{\pm i\}_i$ or $\{\pm i\}_i$) and symmetric exponential ($\{0\} \cup \{\pm 2^i\}_i$) due to their hardware suitability (multiplications can be implemented as bit-shifts). Less common but also used are trained symmetric or arbitrary quantization levels ($\{\pm\alpha_i\}_i$ or $\{\alpha_i\}_i$). Typically, the weights of the convolutional and linear layers are quantized except for the first and last layers in the network, since quantizing these has been shown to have a much stronger impact on the final accuracy than that of the other layers . As in most networks the convolutional and linear layers are followed by batch normalization layers, any linear scaling of the quantization levels has no impact on the optimization problem.

Mixed-integer non-linear programs such as (2) are NP-hard and practical optimization algorithms trying to solve it are only approximate. Most previous works approach this problem by means of annealing a smoothed multi-step function applied to underlying non-quantized weights (and clipping the gradients) or by quantizing the weights in the SGD's forward pass and introducing proxy gradients in the backward pass (e.g., the straight-through estimator (STE)) to allow the optimization to progress despite the gradients being zero almost everywhere. Recently, (Leng et al., 2018) proposed to use the *alternating direction method of multipliers (ADMM)* to address this optimization problem with promising results. However, their method requires a non-standard gradient descent optimizer, thus preventing simple integration into commonly used deep learning toolkits and thereby wide-spread adoption.

3.1 Random Partition Relaxation Algorithm

For RPR, we propose to approach the MINLP through alternating optimization. Starting from continuous values for the parameters in $\mathbb{W}_q$, we randomly partition $\mathbb{W}_q$ into $\mathbb{W}_q^{\text{constr}}$ and $\mathbb{W}_q^{\text{relaxed}}$ for some specified freezing fraction (FF), e.g. $\text{FF} = \frac{\#\mathbb{W}_q^{\text{constr}}}{\#\mathbb{W}_q} = 90\%$. The parameters in $\mathbb{W}_q^{\text{constr}}$ are quantized to their closest value in $\mathbb{L}$ while those in $\mathbb{W}_q^{\text{relaxed}}$ keep their continuous value, which is updated according to

$$\hat{\boldsymbol{w}}_q^{\text{relaxed}}, \hat{\boldsymbol{w}}_c = \underset{\boldsymbol{w}_q^{\text{relaxed}}, \boldsymbol{w}_c}{\arg\min}\ f(\boldsymbol{w}_q^{\text{constr}}, \boldsymbol{w}_q^{\text{relaxed}}, \boldsymbol{w}_c). \tag{3}$$

This allows the relaxed parameters to co-adapt to the constrained/quantized ones. This step is repeated, alternating between optimizing other randomly relaxed partitions of the quantized parameters (cf. Figure 1. As the accuracy converges, FF is increased until it reaches 1, at which point all the constrained parameters are quantized.

The non-linear program (3) can be optimized using standard SGD or its derivatives like Adam, RMSprop, ... We have experimentally found performing gradient descent on (3) for one full epoch before advancing to the next random partition of $\mathbb{W}_q$ to converge faster than other configurations. Note that $\boldsymbol{w}_q^{\text{constr}}$ is always constructed from the underlying continuous representation of $\boldsymbol{w}_q$. We also initialize $\boldsymbol{w}_q^{\text{relaxed}}$ to the corresponding continuous-valued representation as well, thus providing a warm-start for optimizing (3) using gradient descent.

3.2 Initialization

Starting with the standard initialization method for the corresponding network has worked well for training VGG-style networks on CIFAR-10 and ResNet-18 on ImageNet. We experimentally observed that smaller freezing fractions FF can be used for faster convergence at the expense of less reliable convergence to a good local optimum.

However, a network can be quantized much faster and tends to reach a better local optimum when starting from a pre-trained network. When convolution and linear layers are followed by a batch normalization layer, their weights become scale-invariant as the variance and mean are immediately normalized, hence we can define our quantization levels over the range $[-1, 1]$ without adding any restrictions. However, the continuous-valued parameters of a pretrained model might not be scaled suitably. We thus re-scale each filter of each layer i to minimize the ℓ_2 distance between the

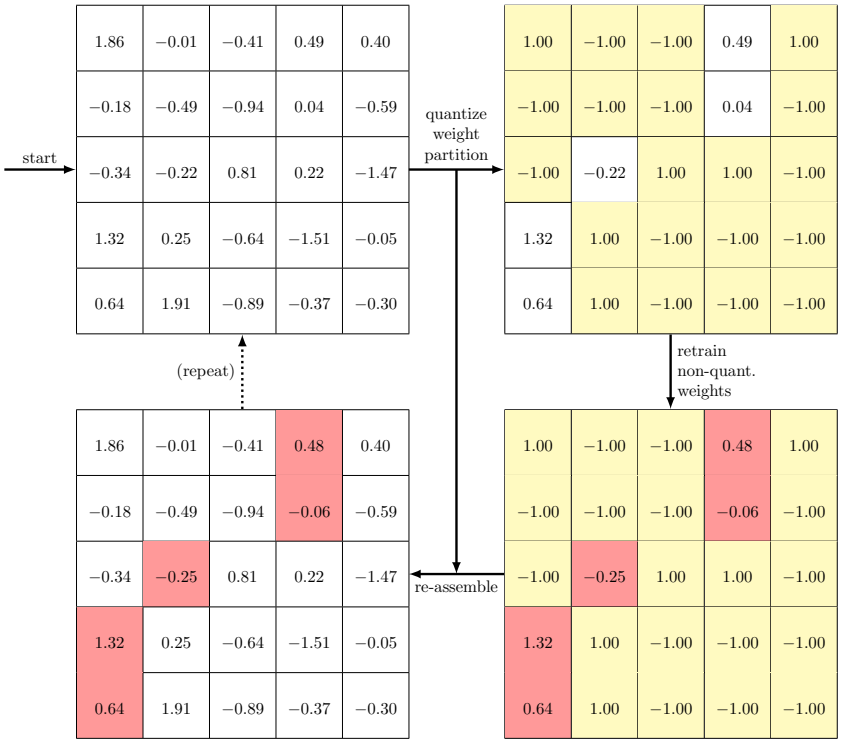

Figure 1: Overview of the Random Partition Relaxation (RPR) algorithm (white: initial parameters in $\boldsymbol{w}_q$, yellow: quantized/constraint-enforced partition $\boldsymbol{w}_q^{\text{constr}}$, red: the optimized values of the relaxed partition $\hat{\boldsymbol{w}}_q^{\text{relaxed}}$).

continuous-valued and the quantized parameters, i.e.

$$\widetilde{\boldsymbol{w}}^{(i)} = \frac{1}{\hat{s}^{(i)}}\boldsymbol{w}^{(i)} \quad \text{with} \quad \hat{s}^{(i)} = \arg\min_{s \geq 0} \|\boldsymbol{w}^{(i)} - s\boldsymbol{w}^{(i)}_{\text{quant}}\|_2 \quad \text{and} \quad \boldsymbol{w}^{(i)}_{\text{quant}} = \arg\min_{\ell \in \mathbb{L}} |\boldsymbol{w}^{(i)} - \ell|. \tag{4}$$

Practically, we implemented (4) using a brute force search over 1000 points spread uniformly over $[0, \max_i |w_i|]$ before locally fine-tuning the best result using the downhill simplex method. The time for this optimization is negligible relative to the overall compute time and in the range of a few minutes for all the weights to be quantized within ResNet-50.

4 Experimental Results

We conducted experiments on ImageNet with ResNet-18, ResNet-50, and GoogLeNet in order to show the performance of RPR by training them as binary weight and ternary weight networks. We refrain from reporting results on CIFAR-10 and with AlexNet on Imagenet as these networks are known to be overparametrized and thus rely on additional regularization techniques not to overfit—this is an irrelevant scenario for resource-efficient deployment of DNNs as a smaller DNN would be selected anyway. Following common practice, we do not quantize the first and last layers of the network. If not stated otherwise, we start from the corresponding pretrained model available through the torchvision v0.4.0 library.

4.1 Preprocessing

The preprocessing and data augmentation methods used in related work vary wildly and from simple image rescaling and croping with horizontal flips and mean/variance normalization to methods with

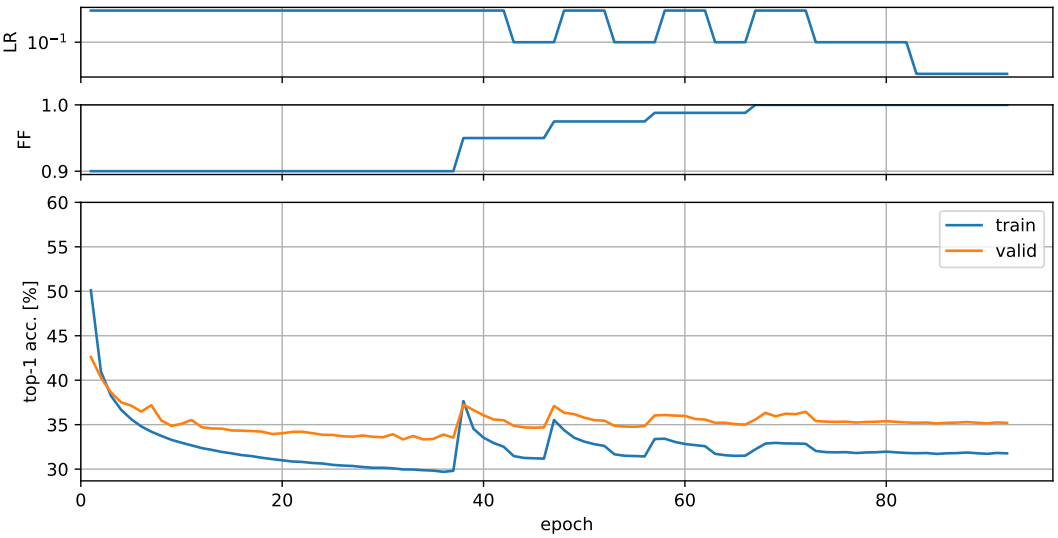

Figure 2: Evolution of the top-1 training and test accuracy together with the schedules for the freezing fraction (FF) and the learning rate (LR) while training GoogLeNet as a ternary weight network.

randomized rescaling, cropping to different aspect ratios, and brightness/contrast/saturation/lighting variations. Consistent with literature, we have found that a quite minimal preprocessing by rescaling the image such that the shorter edge has 256 pixels followed by random crops of 224×224 pixels and random horizontal flips showed best results. During testing, the same resizing and a 224×224 center crop were applied. We observed simpler preprocessing methods working better: this is expected as the original networks' capacities are reduced by the strong quantization, and training the network to correctly classify images sampled from a richer distribution of distortions than that of the original data takes away some of the capacity of the network.

4.2 Hyperparameter Selection & Retraining Time

We trained the networks using the Adam optimizer with initial learning rates identical to the full-precision baseline models (10^{-3} for all models). During an initial training phase we use a freezing fraction $\mathrm{FF} = 0.9$ until stabilization of the validation metric. We proceed with $\mathrm{FF} = 0.95, 0.975, 0.9875, 1.0$. Each different FF was kept for 15 epochs, always starting with the initial learning rate and reducing it by $10\times$ after 10 epochs at the specific FF. After reaching $\mathrm{FF} = 1.0$, the learning rate is kept for 10 cycles each at $1\times$, $0.1\times$, and $0.01\times$ the initial learning rate. An example of a freezing fraction and learning rate schedule is shown in Figure 2.

In practice, quantizing a network with RPR requires a number of training epochs similar to training the full-precision model. This is shown for the quantization of GoogLeNet to ternary weights in Figure 2. The quantization with $\mathrm{FF} = 0.9$ requires 37 epochs followed by 45 epochs of iteratively increasing FF before a final phase of optimizing only the continuous parameters for 30 additional epochs.

4.3 Results & Comparison

We provide an overview of our results and a comparison to related work in Table 1. For ResNet-18, our method shows similar accuracy to the ADMM-based method, clearly outperforming other methods such as the XNOR-net BWN, TWN, and INQ. As discussed before, the ADMM algorithm requires an optimization procedure that is not a simple variation of SGD and has thus not yet found widespread adoption.

A higher accuracy than RPR is achieved by TTQ with an enlarged network ($2.25\times$ as many parameters) and by *Quantization Networks*. Both methods however, introduce trained quantization levels with dire consequences for hardware implementations: either as many multipliers as in full-

Table 1: ImageNet experiments

Model	Method⋆		Levels†	Accuracy [%] (top-1/top-5)
ResNet-18	baseline	torchvision v0.4.0	full-prec.	69.76/89.08
ResNet-18	QN	(Yang et al., 2019)	5: $\{\alpha_i\}_i$	69.90/89.30
ResNet-18	ADMM	(Leng et al., 2018)	5: $\{0\} \cup \{\pm 2^i\}_i$	67.50/87.90
ResNet-18	LQ-Nets	(Zhang et al., 2018)	4: $\{\pm\alpha_i\}_i$	68.00/88.00
ResNet-18	QN	(Yang et al., 2019)	3: $\{\alpha_1, \alpha_2, \alpha_3\}$	69.10/88.90
ResNet-18+‡	TTQ	(Zhu et al., 2017)	3: $\{\alpha_1, 0, \alpha_2\}$	66.60/87.20
ResNet-18	ADMM	(Leng et al., 2018)	3: $\{-1, 0, 1\}$	67.00/88.00
ResNet-18	INQ	(Zhou et al., 2017)	3: $\{-1, 0, 1\}$	66.00/88.00
ResNet-18+‡	TWN	(Li et al., 2016)	3: $\{-1, 0, 1\}$	65.30/86.20
ResNet-18	TWN	(Li et al., 2016)	3: $\{-1, 0, 1\}$	61.80/84.20
ResNet-18	**RPR (ours)**		3: $\{-1, 0, 1\}$	**66.31/87.84**
ResNet-18	ADMM	(Leng et al., 2018)	2: $\{-1, 1\}$	64.80/86.20
ResNet-18	XNOR-net BWN	(Rastegari et al., 2016)	2: $\{-1, 1\}$	60.80/83.00
ResNet-18	**RPR (ours)**		2: $\{-1, 1\}$	**64.62/86.01**
ResNet-50	baseline	torchvision v0.4.0	full-prec.	76.15/92.87
ResNet-50	ADMM	(Leng et al., 2018)	3: $\{-1, 0, 1\}$	72.50/90.70
ResNet-50	TWN	(Li et al., 2016)	3: $\{-1, 0, 1\}$	65.60/86.50
ResNet-50	**RPR (ours)**		3: $\{-1, 0, 1\}$	**71.83/90.28**
ResNet-50	ADMM	(Leng et al., 2018)	2: $\{-1, 1\}$	68.70/88.60
ResNet-50	XNOR-net BWN	(Rastegari et al., 2016)	2: $\{-1, 1\}$	63.90/85.10
ResNet-50	**RPR (ours)**		2: $\{-1, 1\}$	**65.14/86.31**
GoogLeNet	baseline	torchvision v0.4.0	full-prec.	69.78/89.53
GoogLeNet	ADMM	(Leng et al., 2018)	3: $\{-1, 0, 1\}$	63.10/85.40
GoogLeNet	TWN	(Li et al., 2016)	3: $\{-1, 0, 1\}$	61.20/84.10
GoogLeNet	**RPR (ours)**		3: $\{-1, 0, 1\}$	**64.86/86.03**
GoogLeNet	ADMM	(Leng et al., 2018)	2: $\{-1, 1\}$	60.30/83.20
GoogLeNet	XNOR-net BWN	(Rastegari et al., 2016)	2: $\{-1, 1\}$	59.00/82.40
GoogLeNet	**RPR (ours)**		2: $\{-1, 1\}$	**62.01/84.83**

⋆ Unless noted otherwise, the ResNet models have Type-B bypasses (with a 1×1 convolution in the non-residual paths on increase of the feature map count).
† Unless noted otherwise, the first and last layers are excluded from quantization.
‡ Modified network: each layer has $2.25\times$ as many weights.

precision networks are required, or the operations are transformed as $\sum_i w_i x_i = \alpha_1 \sum_i \mathbb{1}_{w_i=\alpha_1} x_i + \alpha_2 \sum_i \mathbb{1}_{w_i=\alpha_2} x_i + \alpha_3 \sum_i \mathbb{1}_{w_i=\alpha_3} x_i$, requiring only very few multiplications but 3 adder trees, thereby increasing the required silicon area for the main compute logic by $\approx 3\times$ with respect to a TWN with a fixed set of quantization levels $\mathbb{L} = \{-1, 0, 1\}$ and can thus be expected to have corresponding effects on energy. For ResNet-50, the results look similar: we achieved accuracies close to the state-of-the-art (i.e., ADMM), but avoiding the added complexity of altering the optimization method beyond a simple derivative of SGD.

For GoogLeNet we surpass the current state-of-the-art, ADMM, by 1.7% top-1 accuracy for binary weights and 1.76% for ternary weights.

5 Conclusion

We have proposed using alternating optimization for training strongly weight-quantized neural networks by randomly relaxing the quantization constraint on small fractions of the weights. We have implemented this method using standard SGD-based optimization. This method improves the state-of-the-art accuracy for binary and ternary weight GoogLeNet and achieves accuracies similar to previous methods on ResNet-18 and ResNet-50 while maintaining easy integrability into existing deep learning toolkits by using standard gradient descent optimization.

REFERENCES

Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, 2018. ISSN 0278-0070. doi: 10.1109/TCAD.2017.2682138.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Adv. NIPS*, pp. 3123–3131, 2015.

Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proc. AAAI*, pp. 3466–3473, 2018. ISBN 9781577358008.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary Weight Networks. In *Adv. NIPS*, 5 2016.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proc. ECCV*, pp. 525–542, 2016.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295–2329, 12 2017. ISSN 0018-9219. doi: 10.1109/JPROC.2017.2761740.

Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization Networks. In *Proc. IEEE CVPR*, 2019.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In *LNCS*, volume 11212, pp. 373–390. 2018. ISBN 9783030012366. doi: 10.1007/978-3-030-01237-3{_}23.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *Proc. ICLR*, 2017.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained Ternary Quantization. In *Proc. ICLR*, 12 2017.