Not All Bits have Equal Value: Heterogeneous Weight Precisions via Trainable Noise Tensors

Abstract

We study the problem of training deep networks while quantizing their parameters into low-precision numeric representations, a setting central to reducing energy consumption and inference time of deployed models. We propose a method that learns different precisions, as measured by bits in numeric representations, for different weights in a neural network, yielding a heterogeneous allocation of bits across parameters. Learning precisions occurs alongside learning weight values, using a strategy derived from a novel framework wherein the intractability of optimizing discrete precisions is approximated by training per-parameter noise magnitudes. Our approach exposes the objective of constructing a low-precision inference-efficient model to the entirety of the training process. Empirical evaluation shows that it is capable of finding highly heterogeneous precision assignments for CNNs trained on CIFAR and ImageNet, improving upon previous state-of-the-art quantization methods. Our improvements further extend to the challenging scenario of learning reduced-precision GANs.

1 Introduction

Scaling up the size of deep neural networks offers returns to accuracy in tasks across a range of application areas, including computer vision and natural language processing. Expanding network depth from tens of layers [21, 41] to a hundred [12, 48] revolutionized image classification and object detection [8, 13]. Following the 340M parameter BERT model [3], the size of state-of-the-art language models has increased by multiple orders of magnitude: GPT-2 with 1.5B [37], GPT-3 with 175B, and recently Megatron-Turing with 530B parameters [42] – more than a 1000-fold increase in size over BERT. Larger models increase the computational cost of inference, posing challenges to their deployment in real-world applications.

To ameliorate these costs, multiple research avenues have explored methods for constructing more compact deep networks while preserving accuracy comparable to larger models. These include, *e.g.,* architectural building blocks with sparser connectivity [16, 34, 50, 15] or efficient dimensionality reduction stages [19]. Another category of approaches shrinks larger models via pruning [10, 29, 27, 7, 22, 40], reducing precision [11, 18, 47], or binarization [17, 38, 32, 35]. Parallel efforts on neural architecture search [53, 33, 25, 30, 26, 45] optimize coarser aspects of model structure by automating exploration of hyperparameters governing network size and configuration of layers.

Yet, these methods fall short of providing a complete framework for producing efficient networks. Pruning, quantization, and binarization are often implemented as heuristic post-processing transformations that operate on a trained network [11, 23] or require special training procedures [17, 38, 49, 52]. Section 2 provides a broader survey. The lack of modular approaches to optimizing the network structure at this finest level of detail complicates potential combination with neural architecture search techniques for selecting the functional form of layers and building blocks at coarser granularity.

Figure 1: **Weight noise as a differentiable proxy for precision.** A learnable magnitude δ scales uniform random noise added to weight w during training. The width of the basin over which it is possible to perturb w without increasing task loss L drives learning of δ. After training, we reduce the bit precision of the numeric representation of w as much as possible, with the constraint of remaining in the $(w - \delta, w + \delta)$ range. *Left:* A random perturbation $\tilde{w}$ increases loss, driving a decrease in δ. *Middle:* Perturbation leaves loss unchanged, driving an increase in δ. *Right:* Noise level δ, and the corresponding implied precision, stabilize when matched to the size of the basin.

We introduce a principled framework for training compact neural networks, which encompasses pruning, quantization, and binarization. Each is a special case of learning a more general heterogeneous mixed-precision architecture, wherein a set of auxiliary parameters govern the bit precision of each weight in the original network architecture. Our approach learns auxiliary parameters simultaneously with learning the original parameters (weights), with training driven entirely by stochastic gradient descent. In addition to modularity, this strategy provides two crucial benefits:

- Unlike schemes which post-process a trained network, our method makes the entirety of the training process aware of the goal of producing a reduced-precision model. Learning weights in the same context as quantization avoids divergence between training and inference scenarios.

- The auxiliary parameters can freely participate in the definition of the loss being optimized during training. We can craft loss terms that approximate actual costs (*e.g.,* in computation or energy usage) of inference as determined by the current configuration of numeric precisions. Combined with the task-specific loss term, we can choose how to trade-off accuracy and efficiency.

Key to our technical approach is correspondence between a notion of noisy weights and numeric precision. Specifically, during training, we instantiate each network weight as a full-precision value plus uniform random noise of a learnable magnitude. These noise magnitudes are precisely the auxiliary parameters. The more noise that can be injected into the value of a particular weight without increasing loss, the lower the precision required to represent that weight. Figure 1 illustrates the intuition behind this correspondence. During training, noise magnitudes serve as a differentiable proxy for discrete bit precisions. Upon completion of training, the learned noise magnitudes dictate replacement of noisy weights with fixed, reduced-precision values. Though related to Bayesian neural networks, the our formulation differs substantially from prior work on Bayesian approaches to network pruning and quantization [28]. The Appendix provide details and principled derivations.

Results in Section 4 show that, on standard benchmarks such as creating weight-quantized CNNs for image classification, our approach outperforms competitors, achieving higher accuracy using fewer bits per parameter. In a more challenging experiment of training GANs, we observe similar improvements. The source of our advantage may span both the ability to simultaneously account for precision and task objectives within the training dynamics, as well as the ability to learn heterogeneous weight precisions at the finest level of granularity.

2   Related Work

Most approaches in the quantization literature assume that a pre-defined precision is given prior to training [51, 49, 9, 47, 18, 31, 2, 24, 11, 5], designating the number of bits used to represent each parameter of the network. The focus of these works is typically to circumvent the obstacle posed by the non-differentiability of the quantization function, as this makes first-order methods inapt to train quantized models due to the lack of meaningful gradients.

DoReFa [51] updates real-valued weights using gradients w.r.t. their quantized forms (*i.e.,* a straight-through estimator [1]), enabling training over quantized parameter values with gradient descent. LQ-Nets [49], PACT [2], and LSQ [5] further flexibilize quantization by also optimizing the quantization step size, which we refer by *quantization scale* throughout our work, *i.e.,* the real value that each bit corresponds to is also learned.

DSQ [9] proposes a smooth approximation to the quantization mapping, which allows weights to be directly trained with SGD without straight-through estimators. SLB [47] introduces per-parameter auxiliary variables that induce distributions over values that quantized weights can take – these variables are trained with gradient descent and a final quantized model is created by approximating each weight's distribution by a point-mass.

More recently, some works on quantization have explored ways to assign different precisions to parameter groups in a neural network [44, 43, 4, 46] – such methods, however, remain a minority in the quantization literature. DNAS [44] uses neural architecture search to map candidate precisions to different parameter groups. HAQ [43] uses reinforcement learning to learn a quantization policy. HAWQ [4] uses second-order information to allocate precisions to different parameter groups.

Closest to our work is BSQ [46], which introduces new variables that are trained with gradient-based methods, which are then used to allocate precisions throughout the network. It operates by first mapping each real-valued weight to a bitstring variable whose length is chosen a-priori. These variables are then iteratively trained with gradient descent and pruned based on their magnitudes, resulting in a decrease in precision whenever a component (a bit) of the bitstring is removed. Finally, the bitstrings are mapped back to weights, and the quantized model undergoes fine-tuning by adopting straight-through estimators for the quantized weights.

Lastly, Fan et al. [6] improve the performance of quantized networks by randomly selecting a subset of weights to be quantized during training while keeping the remaining parameters in their original format. While our approach also uses randomness to improve quantization in some fashion, note that the form of noise samples, along with how and why they are used, are vastly different.

3   Method

3.1   Allocating Precisions by Optimizing Noise

We consider the problem of allocating a precision p (a positive integer which represents the number of bits) to each weight w of a network under a total precision budget (the total number of bits that can be used to represent the model). If we let $q(w, p)$ denote the quantization of w with p many bits, then our goal is to minimize $L(q(w, p))$ subject to $p \leq C$, where C is our total budget measured in bits and L captures the loss incurred by the weights – typically the cross-entropy loss over a training set.

Although our method is the result of a rigorous sequence of approximations to the objective described above, it admits an intuitive description and motivation that relies on a connection between quantization and perturbations. More specifically, if we assume that a weight w can be perturbed in any direction by at least ϵ without degrading the performance of the network, then we can safely represent w with p bits as long as the quantization error does not exceed ϵ – that is, $|w - q(w, p)| \leq \epsilon$. We call our method SMOL : **S**earching for **M**ixed-Precisions by **O**ptimizing **L**imits for Perturbations.

In other words, if we know the largest perturbation that each weight admits (its *perturbation limit*) without yielding performance degradation, then we can easily assign a precision to each weight: it suffices to choose the smallest number of bits such that the quantization error falls below the perturbation limit of the corresponding weight. Our method works by estimating the perturbation limit of each parameter to be quantized, which is achieved by directly optimizing the magnitude of the weight perturbations through a novel loss function.

Once the perturbation limit of each weight has been estimated through optimization by our method, we assign per-weight precisions by mapping each perturbation limit to a number of bits.

More specifically, for a model with weight parameters w and training loss L, our method first introduces new parameters s of the same shape as w, and at each iteration t aims to update w, s to decrease an augmented loss function:

$$L\left(w + \sigma(s) \odot \epsilon^{(t)}\right) + \lambda \left\| \log_2(1 + e^{-s}) \right\|_1, \tag{1}$$

where $\epsilon^{(t)}$ has the same size as s and whose components are independently sampled from an uniform distribution over the interval $[-1, +1]$, and λ controls the trade-off between performance and representational cost – a larger value for λ will place more importance on using less bits to represent weights compared to achieving small loss. Note that the noise samples $\epsilon^{(t)}$ are independent across different iterations, *i.e.,* we draw a new sample $\epsilon^{(t)}$ at each iteration t. Here, the term $\sigma(s)$ represents the perturbation limit estimates of the weights w: the distance between w and $w + \sigma(s) \odot \epsilon^{(k)}$ is at most $\sigma(s)$ since ϵ^k is between -1 and $+1$.

A key aspect of our proposed loss function lies in the fact that it is fully differentiable w.r.t. to s, hence gradient-based methods like SGD and Adam can be applied off-the-shelf to optimize both the original weights w and the auxiliary variables s. This enables the parameters s to be seen as being part of the model itself, allowing for our method to be easily combined with higher-order optimizers, neural architecture search, and other algorithms that operate on networks.

After training, we map the final values of s to precisions

$$p = 1 + \left\lceil \log_2(1 + e^{-s}) \right\rceil, \tag{2}$$

where $\lceil \cdot \rceil$ outputs integers *e.g.,* rounding or truncation.

The resulting tensor p will have the same shape as w, and its components represent the number of bits assigned to each parameter in w. Having allocated a precision to each weight parameter, we can discard the auxiliary variables s and use quantized weights $w_q = q(w, p)$ to compute the model's predictions instead of applying perturbations. This will typically lead to non-negligible changes in the model's activations, which we circumvent by fine-tuning the model by further optimizing w to minimize the training loss. Following prior work, we use straight-through estimators to optimize w since q is non-differentiable, *i.e.,* we set $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w_q}$ and perform gradient descent on w.

Figure 2: Multidimensional example showing how different precisions should be assigned to each parameter depending on their perturbation limits.

Lastly, note that inverting the $s \rightarrow p$ mapping offers a way to initialize s given a desired initial precision p_{init}, more specifically we set $s_{init} = -\ln\left(2^{p_{init}-1} - 1\right)$. For example, adopting an initial precision $p_{init} = 8$ results in $s_{init} = -\ln(2^{8-1} - 1) = -\ln(127)$, and the perturbation limit estimate will start as $\sigma(s_{init}) = 2^{1-p_{init}} = 2^{-7}$ for all weights w of the network.

3.2 Zero Precision Allocation

A fundamental limitation of having each i'th bit map to either $+2^{1-i}$ or -2^{1-i} is that zero weights cannot be represented: the possible values that a 1-bit weight can assume are $\{-1, +1\}$, for a 2-bit weight $\{-1.5, -0.5, +0.5, +1.5\}$, and moreover a quantized weight cannot assume the value 0 regardless of its precision. However, in our setting – where we can assign a different precision to each weight – we can directly re-define the quantization function q to map any w to 0 whenever $p = 0$, hence introducing the notion of *zero-precision weights*. This is analogous to the procedure of assigning zero precision to a filter in BSQ, which leads to the corresponding convolution to be completely skipped when computing a model's outputs.

Algorithm 1 SMOL

1: **procedure** SMOL $(w \in \mathbb{R}^d, L : \mathbb{R}^d \to \mathbb{R}, \lambda \in \mathbb{R}, p_{init} \in \mathbb{N})$
2:     Instantiate parameters $s \in \mathbb{R}^d$ and initialize $s_{init} = -\ln\left(2^{p_{init}-1} - 1\right)$
3:     **for** $t = 1 \to T_1$ **do**
4:         Sample $\epsilon^{(t)} \sim \mathcal{U}^d(\pm 1)$
5:         Compute $\mathcal{L}(w, s) = L(w + \sigma(s) \odot \epsilon^{(t)}) + \lambda \| \log_2(1 + e^{-s}) \|_1$ and $\nabla_{w,s}\mathcal{L}(w, s)$
6:         Update w and s using $\nabla_w\mathcal{L}(w, s)$ and $\nabla_s\mathcal{L}(w, s)$
7:         Clip w to $\pm(2 - \sigma(s))$
8:     **end for**
9:     Set $p = 1 + \text{round}(\log_2(1 + e^{-s}))$
10:     ZPA (Optional): If $|w| < |w - q(w, p)|$ then set $p = 0$ (element-wise)
11:     **for** $t = T_1 \to T_2$ **do**
12:         Compute $L(w_q)$ and $\nabla_{w_q}L(w_q)$, where $w_q = q(w, p)$
13:         Update w using $\nabla_{w_q}L(w_q)$ instead of $\nabla_w L(w_q)$
14:     **end for**
15: **end procedure**

We propose to assign zero precision to a weight w whenever $|w| \le |w - q(w, p)|$. Since the quantization function q maps w to the closest value that is representable with p bits, whenever a weight's precision is set to zero due to our proposed strategy, the induced quantization error is guaranteed not to increase. This follows since prior to changing p the quantization error is $|w - q(w, p)|$; but once we set $p = 0$, it becomes $|w - q(w, 0)| = |w - 0| = |w|$, which cannot lead to an increase since the condition is precisely $|w| \le |w - q(w, p)|$. For example, $q(w = 0.2, p = 2) = 0.5$ since it is the value in $\{-1.5, -0.5, +0.5, +1.5\}$ that is closest to $w = 0.2$, while $q(w = 0.2, p = 0) = 0$ following our re-definition of q yields a quantization error of $|0.2 - 0| = 0.2$. On the other hand, for $p = 2$ we have $|0.5 - 0.2| = 0.3$. Hence, assigning zero precision in this case not only frees up 2 bits but also decreases the quantization error by 0.1.

One advantage of adopting this procedure to assign zero precisions to weights is that it only changes q and hence the quantized network, therefore not affecting the procedure described in the previous section. In other words, one can allocate precisions to weights by optimizing perturbations as described previously, and then quantize the model separately with and without zero precision allocation, resulting in two quantized models with different average precisions, hence obtaining two networks after training the auxiliary variables s only once. The notion of zero precisions also unify quantization and pruning, since in practice assigning zero precision is equivalent to pruning the corresponding weight.

4   Results

We evaluate our method in the tasks of quantizing CNNs trained for image classification and generation. For all experiments, we adopt an initial precision $p_{init} = 8$, which corresponds to initializing each component of the new parameter s as $-\ln(2^7 - 1) \approx -4.84$. We also use the floor operation to map real-valued precisions to integers: based on our preeliminary experiments a more aggressive rounding operation rarely results in performance degradation, while at the same time yielding more compact models.

For all experiments we train the auxiliary parameters s with Adam [20], using the default learning rate of 10^{-3} and no weight decay – all its other hyperparameters are set to their default values. We run SMOL with and without Zero Precision Allocation: allowing for zero precisions results in more compact models, and we use SMOL * to denote results with zero precision allocation.

To compare different methods we measure the performance and the size of the resulting network. We report the average number of bits required to represent the model (the sum of all weight's precisions divided by the number of weights), which we refer to as 'average bpp', along with the compression ratio relative to a full-precision model, which equals to 32 divided by the average bpp.

Table 1: Performance of different quantization methods on ResNet-20 models trained on CIFAR-10. * represents results with Zero Precision Allocation.

| Method | ResNet-20 | | |
	Average Bpp ↓	Compression Ratio ($\times$) ↑	Test Accuracy (%)
LQ Baseline	32.0	1.0	92.1
BSQ Baseline	32.0	1.0	92.6
LQ-Nets	1.0	32.0	90.1
DSQ	1.0	32.0	90.2
SLB	1.0	32.0	90.6
LQ-Nets	2.0	16.0	91.8
SLB	2.0	16.0	92.0
SMOL *	1.7	18.8	92.6
SMOL *	1.3	24.6	91.5
LQ-Nets	3.0	10.7	92.0
BSQ	2.7	11.8	91.9
SMOL	2.5	12.8	92.8
SMOL	2.1	15.2	92.5
SLB	4.0	8.0	92.1
BSQ	3.2	10.0	92.8
SMOL	2.8	11.4	93.0

4.1 ResNet-20 on CIFAR-10

We compare SMOL against different quantization methods on the small-scale CIFAR-10 dataset. Following prior work, we use ResNet-20 models to evaluate our method's performance in terms of how aggressively it can quantize weights without degrading the model's test accuracy.

Since BSQ is the most recent method in our comparison and is also the most related to ours, we adopt most of its experimental settings whenever a comparison is possible. More specifically, our training pipeline for ResNet-20 is similar to the one described in BSQ. On CIFAR-10, we only consider results with full-precision activations, both for our and prior methods. This leads to a more direct comparison – since activation quantization is treated differently across prior works, we isolate each method's performance to its ability to quantize parameters.

We use the same data augmentation procedure for all experiments: at training time images are first padded with 4 pixels and then a random 32×32 crop is extracted. Lastly, images are flipped horizontally with probability = 0.5.

ResNet-20: Unlike BSQ, training is done from scratch instead of fully pre-training the model prior to the precision allocation process. We train for a total of 650 epochs, a considerably smaller budget than BSQ's 1000 total epochs. We allocate the last 300 epochs to fine-tuning, hence the per-parameter precisions are trained for the first 350 epochs and then kept fixed.

Following BSQ, the shortcut connections of our ResNet consist of 1×1 convolutions followed by batch norms, and we set all batch norm layers along with the shortcut convolutions to full-precision while quantizing the remaining parameters. We use SGD with a momentum of 0.9 and an initial learning rate of 0.1, which is decayed at epochs 250, 500, and 600, and adopt a batch size of 128 and a weight decay of 10^{-4}.

Results are reported in Table 1. SMOL comfortably outperforms BSQ and other competing methods when quantizing ResNet-20 models, achieving 92.5% test accuracy with 2.1 bpp: a 0.6% higher than BSQ while also saving 0.6 bpp. For models with similar precisions (*e.g.,* networks with 2.7 and 2.8 bpp), SMOL yields 1.1% higher performance than BSQ, also outperforming the full-precision baseline. When enabling zero-precision allocation, SMOL is able to push compression to extreme levels while preserving the model's generalization, yielding a network with 1.7 bpp and a test accuracy of 92.6%, matching the full-precision baseline.

Table 2: Performance of different quantization methods on ResNet-18 and ResNet-50 models trained on ImageNet. * represents results with Zero Precision Allocation.

Method	ResNet-18			ResNet-50		
	Average Bpp ↓	Compression Ratio (×) ↑	Test Accuracy (%)	Average Bpp ↓	Compression Ratio (×) ↑	Test Accuracy (%)
Baseline	32.0	1.0	69.6	32.0	1.0	76.0
SLB Baseline	32.0	1.0	69.3			
BSQ Baseline				32.0	1.0	76.1
SLB	2.0/4	16.0	67.5			
SMOL *	2.3/4	7.6	69.7	2.2/4		73.9
LQ-Nets	3.0/3	10.7	68.2	3.0/3	10.7	74.2
BSQ				3.3/4	9.7	75.2
BSQ				3.7/4	8.6	75.3
SMOL *	2.8/4	7.6	70.0			
LQ-Nets	4.0/4	8.0	69.3	4.0/4	8.0	75.1
DSQ	4.0/4	8.0	69.6			
SMOL	4.2/4	7.6	70.2			
SMOL	4.5/4	7.1	70.6			

4.2 ResNet-18 and ResNet-50 on ImageNet

For the large-scale ImageNet classification task, we quantize ResNet-18 and ResNet-50 models which allow for comparisons against LQ-Nets, DSQ, SLB, and BSQ. We use different training budgets for the two models, given that ResNet-50 is considerably larger than ResNet-18. For both networks we train the weight parameters with SGD, a momentum of 0.9, a weight decay of 10^{-4} and a batch size of 256 which is distributed across 4 GPUs. We follow LQ-Nets in terms of data augmentation.

We train ResNet-18 for a total of 180 epochs: the first 120 are used for precision training and the last 60 for fine-tuning, and SGD has an initial learning rate of 0.1 which is decayed by 10 at epochs 45, 90, 150, and 165. ResNet-50 models are trained for 90 epochs: 60 of precision training with SMOL followed by 30 for fine-tuning, with an initial learning rate of 0.1 which is decayed by 10 at epochs 30, 65, and 75.

4.3 Image Generation with DCGAN

We train a DCGAN [36] model on the CIFAR-10 dataset to perform unconditional image generation at resolution of 32×32. Generator is implemented as four transposed convolutional layers with batchnorm and ReLU activation. Discriminator consists of four strided convolutional layers with batchnorm and LeakyReLU activation. We only apply low-precision training on Generator, which is used for deployment.

Full-precision model is trained with the binary cross-entropy(BCE) loss for 30 and 160 epochs, using Adam with learning rate = 2e-4 and (β_1, β_2) = (0.5, 0.999). We compare our method with BSQ by generalizing settings from classification (e.g., reparameterization, regularization) to generator updating. For BSQ, we first train a full precision DCGAN for 30 epochs, quantize it to 8-bit precision and conduct 100 epochs of bit learning, where pruning interval is 10 and trade-off hyperparameter is 2e-3. We then fine-tune BSQ for 30 epochs, with a $10\times$ decayed learning rates. Similarly, ours train for 160 epochs in total, with 30 epochs fine-tuning. We randomly generate 10,000 samples for all methods, each assessed with Inception Score(IS) [39] and Fréchet Inception Distance (FID) [14].

As shown in Table 3, our method consistently outperforms BSQ, with higher generation quality and lower bits, demonstrating the generalization capability to the challenging GAN quantization task. We drastically improves 3.9-bit BSQ's FID from 37.1 to 29.9 with only 3.5 bits. For BSQ at 2.8 bits, SMOL * achieves 31.1 of better FID with only 1 bit. Even evaluated at extreme low bits of 0.2, our method still generates images with good quality.

Table 3: Image Generation on CIFAR-10.

Method	DCGAN		
	Average Bpp	Inception Score ↑	FID ↓
Baseline (30 epochs)	32.0	4.70	38.6
Baseline (160 epochs)	32.0	5.67	25.8
BSQ	2.8	4.85	38.3
SMOL *	0.2	4.75	35.6
SMOL *	0.5	5.03	34.1
SMOL *	1.0	5.01	31.1
SMOL	1.7	5.02	34.1
SMOL	2.5	4.93	35.3
BSQ	3.9	4.95	37.1
SMOL *	3.5	5.29	29.9
SMOL *	3.9	5.07	31.1

Figure 3: Image generations with a DCGAN trained on CIFAR-10, quantized with BSQ and SMOL.

5 Discussion

With advances in hardware design and significant overparameterization of recent deep networks, learning compact model representations that can be leveraged by specialized hardware has become a key problem – one that is not restricted to the machine learning and systems communities. Our work goes a step further in terms of network quantization by proposing SMOL, a modular and principled method that is able to learn heterogeneous precisions at the finest granularity, where different weights of the same model can be represented in distinct precisions regardless of how the weights are connected in the network's topology.

Moreover, SMOL operates by introducing auxiliary variables to the target model, thus being agnostic to the network's topology, the choice of optimizer, or the underlying training setting, and hence it can be easily combined with other algorithms such as neural architecture search. Our procedure to allocate precisions lies in the optimization of a trade-off between the model's performance and its cost – a cost function that can be chosen according to the underlying application or the user's concern *e.g.,* computational or environmental cost.

Finally, our method offers an unification of pruning, binarization, and quantization, while at the same time achieving state-of-the-art results in image classification and image generation with quantized models. Our results show that adopting heterogeneous precisions yields highly compact models that not only well, but can also result in significant computational advantages when deployed on specialized hardware.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv preprint arXiv:1805.06085*, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. 2019.

[5] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. 2020.

[6] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. Training with quantization noise for extreme model compression. 2021.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[9] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *ICCV*, 2019.

[10] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015.

[11] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. CondenseNet: An efficient DenseNet using learned group convolutions. *CVPR*, 2018.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. volume 18, pages 187:1–187:30, 2017.

[19] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *arXiv:1602.07360*, 2016.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[22] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham M. Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, 2020.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *ICLR*, 2017.

[24] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. 2017.

[25] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

[27] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[28] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *NeurIPS*, 2017.

[29] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018.

[30] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.

[31] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[32] Jorn W. T. Peters and Max Welling. Probabilistic binary neural networks. volume abs/1809.03368, 2018.

[33] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

[34] Ameya Prabhu, Girish Varma, and Anoop M. Namboodiri. Deep expander networks: Efficient deep networks from graph theory. *ECCV*, 2018.

[35] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *CVPR*, 2020.

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[37] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[38] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[39] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016.

[40] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *NeurIPS*, 2020.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[42] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zheng, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[43] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. 2019.

[44] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search. *arXiv preprint arXiv:1812.00090*, 2018.

[45] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

[46] Huanrui Yang, Lin Duan, Yiran Chen, and Hai Li. Bsq: Exploring bit-level sparsity for mixed-precision neural network quantization. *arXiv preprint arXiv:2102.10462*, 2021.

[47] Zhaohui Yang, Yunhe Wang, Kai Han, Chunjing Xu, Chao Xu, Dacheng Tao, and Chang Xu. Searching for low-bit weights in quantized neural networks. In *NeurIPS*, 2020.

[48] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[49] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, 2018.

[50] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[51] 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. *arXiv preprint arXiv:1606.06160*, 2016.

[52] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian D. Reid. Training quantized neural networks with a full-precision auxiliary module. In *CVPR*, 2020.

[53] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.