

QBitOpt: Fast and Accurate Bitwidth Reallocation during Training

Jorn Peters, Marios Fournarakis, Markus Nagel, Mart van Baalen, Tijmen Blankevoort
Qualcomm AI Research*
Amsterdam, The Netherlands

{jpeters,mfournar,markusn,mart,tijmen}@qti.qualcomm.com

Abstract

Quantizing neural networks is one of the most effective methods for achieving efficient inference on mobile and embedded devices. In particular, mixed precision quantized (MPQ) networks, whose layers can be quantized to different bitwidths, achieve better task performance for the same resource constraint compared to networks with homogeneous bitwidths. However, finding the optimal bitwidth allocation is a challenging problem as the search space grows exponentially with the number of layers in the network. In this paper, we propose *QBitOpt*, a novel algorithm for updating bitwidths during quantization-aware training (QAT). We formulate the bitwidth allocation problem as a constraint optimization problem. By combining fast-to-compute sensitivities with efficient solvers during QAT, *QBitOpt* can produce mixed-precision networks with high task performance guaranteed to satisfy strict resource constraints. This contrasts with existing mixed-precision methods that learn bitwidths using gradients and cannot provide such guarantees. We evaluate *QBitOpt* on ImageNet and confirm we outperform fixed-precision methods. We also achieve comparable accuracy to other mixed-precision methods, while always meeting the exact resource constraint without the need for hyper-parameter search over regularization strength.

1. Introduction

Neural networks have driven some of the most significant advances in artificial intelligence. However, searching for better-performing neural networks has increased their resource requirements substantially. This is most notable in transformer models in computer vision [12, 39] and natural language understanding [34, 5]. Deploying these ever-growing models on resource-constrained devices, such as mobile phones, embedded systems, and IoT devices, remains a challenge. Quantization has been proven to be a

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

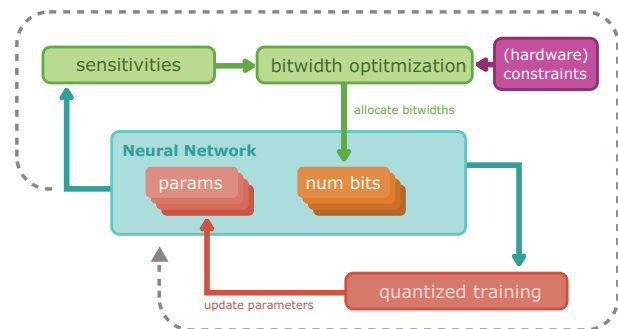


Figure 1: QBitOpt uses a quantization sensitivity metric to infer bitwidths while taking (hardware) constraints into account. The bitwidth allocation is used within a QAT training loop.

very effective method of addressing this constraint by reducing the memory and computational cost of neural networks without sacrificing task performance [22, 26, 29]. This is achieved by compressing weights and activations to a more efficient low bitwidth fixed-point representation. Deploying quantized neural networks on devices is also becoming easier as an increasing number of devices support efficient low-precision operations.

Various methods for neural network quantization have emerged in recent years. Broadly, we can distinguish them into two main groups: *post-training quantization* (PTQ) [31, 43, 28, 15, 27, 16], and *quantization-aware training* (QAT) [24, 14, 25, 30, 40, 23]. In PTQ, we take a pre-trained neural network as input and infer the quantization setting that maximizes task performance without access to labeled data or the original training pipeline. In contrast, in QAT, we simulate quantization during training and allow the network parameters to adjust to quantization noise. In both cases, the target quantization bitwidth is commonly determined beforehand and is set homogeneously over the entire network. A crucial drawback of this approach is that the homogeneous bitwidth can only be as low as the network’s most sensitive layer allows.

However, not all layers in a neural network are equally sensitive to quantization. Hence, it is beneficial to quantize some layers at a higher bitwidth than others. For example, it is common in quantization literature [14, 23, 13, 30] to keep the first and last layer of a neural network at a higher bitwidth than the rest of the network because it improves accuracy with only an incremental increase in resource requirements. Allowing for a heterogeneous bitwidth allocation across the layers of a neural network is called *mixed-precision quantization* (MPQ). The main challenge of MPQ algorithms is to determine the optimal bitwidth per layer that achieves the highest task performance while minimizing resource requirements. In general, these are opposing objectives leading to a trade-off. What makes the MPQ particularly hard is that the search space grows exponentially in the number of neural network layers.

Since the bitwidth allocation problem is a trade-off between neural network resources and task performance, there will be many Pareto optimal solutions to the problem. However, in many real-world applications, the specific deployment target and associated resource constraints are often known beforehand. Assuming this, the MPQ bitwidth allocation problem becomes a constrained single objective optimization problem. Leveraging this observation, we propose QBitOpt, a novel method for inferring near-optimal bitwidth allocations during QAT. We achieve this by measuring the *sensitivity* of each layer to quantization noise and assigning them bitwidths that (1) satisfy the total resource requirements constraints and (2) minimize the total network quantization sensitivity. Our method stands out from previous mixed-precision work in the following ways:

- It is guaranteed to satisfy resource constraints. Most existing methods rely on hyperparameter search to balance the accuracy/resource constraints without guarantees.
- By formulating the bitwidth allocation problem as a constrained convex optimization problem, our method scales to large networks.
- We integrate optimization-based bitwidth allocation with existing quantization-aware training methods and perform competitively competing mixed-precision methods on ImageNet under average bitwidths constraints.
- We show that updating the bitwidth allocation during training is crucial for optimal performance and that it outperforms common post-training bitwidth allocation followed by quantization-aware fine-tuning.

2. Neural network quantization

The purpose of neural network quantization is to reduce the resource requirements and improve the latency of neural

network inference while maintaining the task performance as much as possible and without changing the original network architecture. To achieve this we quantize the parameters and activations of a network to a quantization grid that is commonly learned during QAT [14, 6, 40]. In this paper, we define the quantization operation as follows:

$$Q(x; z, \delta, b) = \delta \cdot \text{clamp} \left(\left\lfloor \frac{x+z}{\delta} \right\rfloor; l(b), u(b) \right) - z. \quad (1)$$

Here x denotes the quantizer input (*i.e.* network parameters or activations), z the zero-point of the quantizer, and δ the quantizer step-size. $u(b)$ and $l(b)$ map the quantizer bitwidth to the upper and lower clamping threshold in the integer domain. These can differ depending on the quantizer specifics (*e.g.* (a)symmetric or (un)signed quantizers). $\lfloor \cdot \rfloor$ denotes the round-to-nearest integer mapping.

A neural network is quantized by a set of K quantizer $\{Q_1, \dots, Q_K\}$, each associated with a different part of the neural network, such as parameters or activations. In this paper, we assume *symmetric uniform quantization* [29] (*i.e.* $z = 0$) for all quantizers in the network. Moreover, for quantizer i , the step-size δ_i is defined in terms of the quantizer bitwidth b_i and a learned range parameter α_i :

$$\delta_i = \alpha_i / (2^{b_i} - 1). \quad (2)$$

For ease of presentation, throughout this paper, we assume unsigned integer quantization. For more details on neural network quantization, we refer the reader to [29, 17].

To enable gradient-based learning during QAT, we need to overcome the fact that the gradient of the step function is zero almost everywhere. A common solution is to approximate the gradient of the quantization function w.r.t. its input with the straight-through estimator (STE) [21, 3]. [14, 25] extend this approach to allow for gradient-based range learning.

2.1. Pseudo-quantization noise

Quantization has been studied in-depth in classical signal processing literature [18]. A common assumption is that the quantization error $\epsilon = x - Q(x)$ is uniformly distributed with zero-mean and a standard deviation of $\delta/\sqrt{12}$, where δ denotes the step-size of the quantizer Q . This assumption can be shown to hold exactly under some conditions [44, 45]. The relationship between quantization noise and quantization is studied in-depth in [46] and, when used in practice, the noise $\epsilon_q = x - Q(x) \sim \mathcal{U}[-\frac{\delta}{2}, \frac{\delta}{2}]$ is referred to as *pseudo-quantization noise* (PQN).

Following the PQN formulation, we define the pseudo-quantization noise quantizer Q_P as follows:

$$Q_P(x; z, \delta, b) = \delta \cdot \text{clamp} \left(\frac{x+z}{\delta} + \epsilon; l(b), u(b) \right) - z. \quad (3)$$

Here $\epsilon \sim [-\frac{1}{2}, \frac{1}{2}]$ and everything else remains the same as in (1). The PQN formulation is an attractive proposition for QAT because it is differentiable and, for this reason, it has been studied by [8, 38]. However, there is no concrete evidence that it outperforms hard quantization with STE.

2.2. Mixed-precision quantization objective

In mixed-precision quantization, we want to solve a multi-objective optimization problem, which involves maximizing the task performance while minimizing resource requirements. Assuming a neural network with parameters Θ , we quantize each part of this network is quantized by a separate quantizer $Q_i \in \{Q_1, \dots, Q_K\}$. For example, one quantizer for parameters and activations per layer. Let $b_i \in \mathbf{b} = [b_1, \dots, b_K]$ denote the bitwidth associated with quantizer Q_i . We denote the quantized forward pass of this neural network, using a specific bitwidth allocation as $Q(\Theta, \mathbf{b})$ and the quantized task loss as $\mathcal{L}(Q(\Theta, \mathbf{b}))$ for an arbitrary task loss \mathcal{L} .

To complete the MPQ task, we introduce a resource cost $\pi(\mathbf{b})$, e.g., memory requirements, CPU cycle counts, or runtime. Throughout this paper, we assume the resource cost (or constraint) is either differentiable or a differentiable relaxation exists. An often used approach to MPQ task is minimizing a scalarized objective [41, 40, 38] similar to:

$$\mathcal{L}_{\text{MPQ}}(\Theta, \mathbf{b}) = \lambda \mathcal{L}(Q(\Theta; \mathbf{b})) + \pi(\mathbf{b}), \quad (4)$$

where $\lambda > 0$ is a scaling factor that weights the relative importance between the task loss and the resource cost. This approach is common because it allows us to learn the bitwidths and network parameters simultaneously, but it suffers from two main drawbacks: (1) it requires searching over values of λ to find a trade-off that is acceptable to the user, and (2) there are no guarantees that the constraint will be met (cf. section 5.3). Our proposed method is designed to mitigate these constraints.

3. QBitOpt

We introduce QBitOpt, our novel method for mixed-precision quantization (MPQ). We previously established that the MPQ objective is a multi-objective optimization problem. As a result, the solution to the scalarized objective from equation (4) will depend on the weighting term λ . In practice, finding the optimal λ between the two objectives that meet real-life resource constraints can be cumbersome and time-consuming. Instead, choosing a strict upper bound on the resources and optimizing the task performance within these bounds would be beneficial.

We adopt this approach for QBitOpt, resulting in the following constrained optimization objective:

$$\begin{aligned} & \text{minimize} \quad \mathcal{L}_Q(\Theta, \mathbf{b}) = \mathcal{L}(Q(\Theta, \mathbf{b})) \\ & \text{subject to} \quad \hat{\pi}(\mathbf{b}) \leq 0 \end{aligned} \quad (5)$$

In the above, $\hat{\pi}(\cdot)$ denotes the resource constraint and \preceq element-wise inequality. This encodes a similar resource cost as $\pi(\cdot)$ in equation (4). However, instead of minimizing the resource constraint, we only aim to satisfy an upper-bound constraint. Note that whereas we describe our method in terms of parameter quantization, this is not an intrinsic limitation of our method. We discuss details of activation quantization in appendix A.

Following the PQN model, we approximate the optimization objective further:

$$\begin{aligned} \mathcal{L}(Q_P(\Theta, \mathbf{b})) &\approx \mathbb{E}_\epsilon [\mathcal{L}(\Theta + \epsilon')], & \epsilon' &\sim \mathcal{U}[-\frac{\delta}{2}, \frac{\delta}{2}], \\ &= \mathbb{E}_\epsilon \left[\mathcal{L}\left(\Theta + \frac{\alpha}{2^b - 1} \epsilon\right) \right], & \epsilon &\sim \mathcal{U}[-\frac{1}{2}, \frac{1}{2}], \end{aligned} \quad (6)$$

where $\delta = [\delta_1, \dots, \delta_{|\Theta|}]$ denotes the step-size associated with each parameter. Moreover, α and \mathbf{b} broadcast to the parameter dimensions. Following our formulation, the step size is a function of both the quantizer range α and the bitwidth \mathbf{b} . The reparametrization in the second equality follows from this. By splitting the optimization objective in (5) over the network parameters and the bitwidth allocation into two separate minimizations, and substituting the result of equation (6), we obtain our QBitOpt optimization objective:

$$\min_{\Theta} \min_{\mathbf{b}} \mathbb{E}_\epsilon \left[\mathcal{L}\left(\Theta + \frac{\alpha}{2^b - 1} \epsilon\right) \right] \quad \text{s.t.} \quad \hat{\pi}(\mathbf{b}) \leq 0. \quad (7)$$

This objective can now be optimized directly using, e.g., the penalty method [47] or the augmented Lagrangian method [47]. However, this would require retraining the neural network to converge multiple times, which is costly. Instead, in QBitOpt, we efficiently solve an approximation to the inner minimization problem and use the solution \mathbf{b}^* to update Θ using common gradient-based QAT techniques.

3.1. Approximate bitwidth minimization

The QBitOpt optimization, as stated in (7), can be solved directly, but doing so may be very costly. To enable gradient-based training of the network parameters Θ , we derive an efficient method for inferring the quantizer bitwidth and solving the bitwidth allocation problem. To this end, we focus on the inner optimization problem of (7):

$$\mathbf{b}^* = \arg \min_{\mathbf{b}} \mathbb{E}_\epsilon \left[\mathcal{L}\left(\Theta + \frac{\alpha}{2^b - 1} \epsilon\right) \right] \quad \text{s.t.} \quad \hat{\pi}(\mathbf{b}) \leq 0. \quad (8)$$

We first approximate (8) using a second-order Taylor approximation w.r.t $\alpha/(2^b - 1)$ around $\mathbf{0}$:

$$\begin{aligned} \mathbf{b}^* &= \min_{\mathbf{b}} \mathbf{h}^\top \left(\frac{\alpha}{2^b - 1} \right)^2, & \mathbf{h}_i &= \nabla^2 \mathcal{L}(\Theta)_{ii} \\ & \text{subject to} \quad \hat{\pi}(\mathbf{b}) \leq 0, \end{aligned} \quad (9)$$

where \mathbf{h} is the diagonal of the Hessian of the neural network. The first-order term cancels because the expectation of the pseudo-quantization noise is zero, and the constant term does not depend on \mathbf{b} . For the full derivation, see appendix B. The obtained objective is similar to the objective in [11]. However, they make an explicit assumption that the pre-trained floating-point network has converged, whereas we reach the same conclusion using only the zero-mean PQN assumption and make no assumptions about the state of the network. This distinction is important when solving (7) because it allows us to solve this inner optimization during QAT regardless of whether the outer optimization over Θ has converged yet.

Depending on the exact choice of resource constraints $\hat{\pi}$, (9) is a convex program and can be solved very efficiently. In section 3.2, we discuss the choice of resource constraints and optimization methods in more detail.

Putting everything together, the QbitOpt optimization procedure consists of the following steps: first, solve (9) to obtain \mathbf{b}^* , and second, perform backpropagation keeping \mathbf{b}^* constant to update step for Θ and α using gradients:

$$\nabla_{(\Theta, \alpha)} \mathbb{E}_{\epsilon} \left[\mathcal{L} \left(\Theta + \frac{\alpha}{2^{b^*} - 1} \epsilon \right) \right]. \quad (10)$$

3.1.1 Quantization Sensitivities

The optimization procedure we established for \mathbf{b} in (9) depends on the diagonal of the Hessian of the neural network. In this optimization problem, the Hessian diagonal captures how sensitive a parameter or a group of parameters is to quantization or perturbations in general. We formally define *sensitivity* as a statistic that quantifies how much the output of a neural network is affected by quantization. The diagonal of the Hessian in (9) is only one type of sensitivity, and one can imagine using other measures of sensitivity. In fact, this could be beneficial as calculating the Hessian diagonal of a neural network is often computationally expensive. This is why we use an approximation to the hessian diagonal called FIT [50]. Another potential shortcoming of the Hessian is that it captures the effect of infinitesimal perturbations, whereas low-bit quantization perturbations may be pretty significant. We leave such an investigation for future work.

FIT is a sensitivity that has been shown to correlate well with the hessian diagonal [50]. For this reason, we use FIT as a drop-in replacement for the more expensive hessian diagonal. It is defined as:

$$\tilde{\mathbf{h}}_i = [\nabla_{\Theta} \mathcal{L}(\Theta)]^2. \quad (11)$$

Note that FIT, similar to the hessian sensitivity, is computed based on the non-quantized neural network. As such, it requires an extra *full precision* forward and backward pass to

compute and update sensitivities. Moreover, since we can only obtain gradients with respect to a subset of the training set, the computed FIT sensitivities are stochastic in nature. To address the stochasticity in the sensitivity estimation and to reduce the computational requirements, we keep a running exponential moving average of FIT sensitivities and only update them every τ iteration. See Appendix D.2 for more details on the choice of the EMA estimator.

So far, we have only focused on the quantization noise that stems from the rounding operator and can be effectively modeled as additive noise with PQN. However, the quantization operator of equation (1) also involves *clipping* of values outside the quantization grid limits. Clipping becomes an integral part of the training during QAT and can lead to considerable accuracy degradation if removed from the quantized model [1]. For this reason, we compute FIT using the clipped model parameters $\hat{\Theta} = \text{clip}(\Theta, \alpha)$.

3.2. Choice of Resource Constraints and Optimization Methods

So far, we have left the resource constraint $\hat{\pi}(\cdot)$ undefined. In this section, we discuss the choices for $\hat{\pi}(\cdot)$ and its implications on the optimization problem. We aim to satisfy two goals: First, the optimization problem must be efficiently solved to be integrated into training, and second, the constraints should express realistic resource constraints. We accomplish the first goal by restricting the constraints to be of the form:

$$\hat{\pi}_i(\mathbf{b}) \leq 0 \quad i = 1, \dots, m \quad (12)$$

$$\rho_j(\mathbf{b}) = 0 \quad j = 1, \dots, p, \quad (13)$$

where the inequality constraint functions $\hat{\pi}_i(\mathbf{b})$ are convex functions and the equality constraint functions $\rho_i(\mathbf{b})$ are affine. Combining constraints of this form with the optimization objective (9) results in a convex program¹ [4]. Such convex programs can be solved efficiently using off-the-shelf convex optimization libraries, such as [9], allowing us to solve the bitwidth allocation problem frequently during training. In section 5.2, we demonstrate that updating bitwidths during training leads to higher task performance.

The convexity constraints of (12) and (13) may appear quite restrictive at first, but a lot of real-life constraints can be expressed this way, such as the *BOPs*, the network's *average bitwidth* or *the network size bits*. Even if the resource constraints are not strictly convex, there is extensive literature on expressing non-convex problems as convex optimization problems [4, Part II].

In our constraint optimization objective so far, we have not restricted the solution \mathbf{b}^* to be a positive integer, which

¹we referred to a single constraint $\hat{\pi}$ up to this point. However, the convex program defined here allows for multiple inequality and equality constraints at the same time. Hence, $\hat{\pi}$ refers to a set of constraints.

is necessary for finding realistic solutions. Including such a constraint turns the optimization problem into an integer program which is, in general, NP-complete [33] and would be infeasible to solve for large networks. However, certain resource constraints admit efficient and specific optimization procedures. For example, the average bitwidth constraint admits the *greedy integer* method presented in Algorithm 2. Alternatively, we can relax the integer constraint by allowing *fractional* bits and solve the optimization problem efficiently using a convex program. We can use this relaxation during the bitwidth update phase of training, where speed is important, and perform a final expensive integer program to infer the final bitwidth allocation before the final fine-tuning stage. We discuss these choices in more detail in ablation studies of section 5.2.

3.3. Integrated QBitOpt QAT pipeline

Algorithm 1 describes our integrated QAT and QBitOpt pipeline. At each training iteration, we update the quantization sensitivities S_q for each quantizer using an exponential moving average, as described in (3.1.1), and perform a back-propagation step keeping the bitwidth \mathbf{b} to update the network Θ and quantization parameters α . For every τ iteration, we update the quantizer bitwidth, solving the convex optimization problem under the specified resource constraint, described in section 3.2. We do not update the bitwidths at every iteration in order to reduce the computational overhead and to give the network parameters enough time to adjust to the new bitwidth allocation. For a high-level schematic, see Figure 1.

Algorithm 1 QBitOpt

Require: Neural network \mathcal{N} , task loss $\mathcal{L}(\cdot)$, resource constraints $\hat{\pi}(\cdot)$, and sensitivity method \mathcal{S}

for iteration $i = 1, \dots, T$ **do**

for quantizer $q = 1, \dots, K$ **do**

$S_q^{(i)} = \gamma \mathcal{S}(\mathcal{L}(\Theta^{(i-1)}))_q + (1 - \gamma) S_q^{(i-1)}$

end for

if $i \bmod \tau == 0$ **then**

$\mathbf{b} = \text{BitOptimizer}(S_1^{(i)}, \dots, S_k^{(i)})$

end if

$\Theta^{(i)} = \Theta^{(i-1)} + \eta \nabla_{\Theta} \mathcal{L}(Q(\Theta^{(i-1)}; \mathbf{b}, \alpha^{(i-1)}))$

$\alpha^{(i)} = \alpha^{(i-1)} + \eta \nabla_{\alpha} \mathcal{L}(Q(\Theta^{(i-1)}; \mathbf{b}, \alpha^{(i-1)}))$

end for

4. Related Work

Neural network quantization has been a topic of interest in the research community for several years due to attractive hardware properties of quantized networks [7, 19, 24, 2, 14]. For recent in-depth surveys on PTQ and QAT topics, we refer the reader to [29] and [17].

Algorithm 2 Greedy integer assignment for average bitwidth

Initialize $\mathbf{b} \leftarrow [1, \dots, 1]$

$\mathcal{B} \leftarrow |\mathbf{b}| \cdot (\beta - 1)$

$\mathcal{S}_q \leftarrow$ sum of param./act. sensitivities for quantizer q

for iteration $1, \dots, \mathcal{B}$ **do**

$q \leftarrow \arg \min_j \mathcal{S}_j [\alpha_j / (2^{b_j} - 1)]$

$b_q \leftarrow b_q + 1$

end for

Several methods for MPQ have been introduced in recent years. In gradient-based MPQ approaches [41, 40, 20, 48], a bitwidth for each quantizer is learned through gradient-based optimization. An extension of gradient-based methods can be found in PQN-based approaches [8, 38]. Using PQN instead of simulated quantization avoids the (biased) STE [21] and allows direct learning of each quantizer’s bitwidth. While these methods are efficient to run, defining a specific resource or accuracy target is impossible. Instead, a search over some regularization hyper-parameter is necessary. Several reinforcement learning techniques have been introduced [42, 35], in which an agent selects quantization policies that balance hardware constraints with accuracy targets. These methods show promising results but require very long training times to achieve good performance.

Our work extends to a different branch of MPQ literature, in which *sensitivity* is used as a statistic to select a bitwidth for each quantizer. In these methods, quantizers with high quantization sensitivity are assigned higher bitwidths. Similarly to our work, the HAWQ line of research [11, 10, 49] defines sensitivity as either the spectral norm or the trace of the Hessian. Their approach significantly differs from ours in how sensitivity is incorporated: due to the large computational requirements of their method, bitwidths are only estimated once, after which the network is iteratively fine-tuned. [51] use a first-order Taylor expansion information to estimate sensitivity during QAT and reduce bitwidths for quantizers for the least sensitive quantizers. Since their method only allows iterative bitwidth reduction, it can never recover from faulty bitwidth assignments. Taking a slightly different approach, [32] measure a quantizer’s sensitivity through signal-to-noise ratio on the network’s output by lowering the bitwidth of a target quantizer. They then use this information to iteratively reduce each quantizer’s bitwidth until some target accuracy or efficiency metric is reached. While the authors show good results, similarly to the HAWQ line of work, this approach is too slow to incorporate in QAT and is only used in PTQ settings. FIT [50] proposes a new sensitivity metric based on the trace of the Empirical Fisher matrix, which efficiently approximates the computationally expensive Hessian sensitivity in HAWQ. We use this sensitivity metric in

QBitOpt to perform bitwidth allocation during QAT efficiently. Lastly, [37] compares several sensitivity metrics for post-training MPQ.

5. Experiments

In this section, we evaluate the effectiveness of QBitOpt by comparing it with other fixed-precision QAT methods and mixed-precision methods from the literature on the ImageNet [36] classification benchmark. We focus on low-bit quantization (4 and 3 bits on average) of efficient networks with depth-wise separable methods that are generally harder to quantize than fully convolutional networks [30, 29].

5.1. Experimental setup

Quantization We follow the example of existing QAT literature and quantize the input to all layers except for the first layer and normalizing layers. In contrast to most existing QAT literature, we quantize *all* layers, including the first and last layer, and let QBitOpt decide the optimal bitwidth for these layers. We quantize weights (per-channel) and activation (per-tensor) using hard quantization and train all network parameters, including the quantization threshold α , using the straight-through estimator, similar to LSQ [14].

Mixed-Precision Our training consists of two phases. In the first phase of *mixed-precision* QAT, we calculate FIT sensitivities while training the quantized network and re-allocate bitwidths every $\tau=250$ training iterations. In the second (or *fine-tuning*) phase, we freeze the obtained bitwidth allocation and fine-tune the remainder of the trainable parameters. This two-phase approach is quite common in the mixed-precision literature [38, 41]. Both phases are balanced in our experiments, and each takes up 50% of the training time unless stated otherwise. We study the effect of this choice in table 4.

Resource Constraint In most experiments, we use the *average bitwidth* constraint across all quantizers in the neural network, including activations and weights. We restrict the bitwidths to integers that are greater or equal to 2 bits and solve the optimization problem with the greedy algorithm 1. The abbreviation X/X_{MP} below reflects this constraint with a target average bitwidth of X .

In table 6, we additionally use the *average bitwidth weighed by tensor size* constraint, which is applied independently for weights and activations:

$$\rho_w(\mathbf{b}_w) = \frac{\sum_i b_i^w \cdot e_i^w}{\sum_i e_i^w}, \quad \mathbf{b}_w \preceq 2, \quad (14)$$

$$\rho_a(\mathbf{b}_a) = \frac{\sum_i b_i^a \cdot e_i^a}{\sum_i e_i^a}, \quad \mathbf{b}_a \preceq 2, \quad (15)$$

Architecture	W/A	\tilde{h}_{FP}	\tilde{h}_{clip}
MobileNetV2	4/4 _{MP}	69.44	69.71
MobileNetV3-Small	4/4 _{MP}	63.47	64.23
	3/3 _{MP}	56.64	57.14

Table 1: QBitOpt using clipped sensitivities \tilde{h}_{clip} and full-precision sensitivities \tilde{h}_{FP} . Validation accuracy (%) on ImageNet.

Architecture	W/A	Fractional	Integer
MobileNetV2	4/4 _{MP}	67.91	69.71
	3/3 _{MP}	65.65	65.65
MobileNetV3-Small	4/4 _{MP}	64.23	64.39
	3/3 _{MP}	57.14	57.14

Table 2: Integer vs. fractional bits. Validation accuracy (%) on ImageNet.

where b_i^w/b_i^a and e_i^w/e_i^a denote the bitwidth and number of elements of the i^{th} layer’s weight/activation, respectively. These results are included for a fair comparison to the NIPQ [38] method, which uses this constraint.

Optimization In all cases, we start from a pre-trained full-precision network (see appendix C.3 for the origin of the checkpoints) and instantiate the weight and activation quantization parameters using MSE range estimation [29]. We train MobileNetV2 and EfficientNet-Lite for 30 epochs with SGD and momentum of 0.9 and MobileNetV3-Small for 40 epochs. More details about the optimization can be found in the appendix C.1.

5.2. Ablation studies

Sensitivities computation In section 3.1.1, we argued that *clipped* network sensitivities are better suited for inferring bitwidth during QAT. The results in table 1 confirm our hypothesis and show that in all cases, using the clipped sensitivities leads to better accuracy. In some cases, such as for MobileNetV3, not clipping during sensitivity computation can result in a substantial accuracy drop.

Fractional vs. integer bitwidth Constraining all bitwidths to natural numbers turns the convex program into an integer program (see section 3.2), which is NP-hard. Here, we experimentally validate and compare our two proposed solutions, the *greedy integer* method and *fractional bitwidth*. In table 2, we show the effect of both optimization methods on the final accuracy. We observe that in most cases, these lead to very similar performance, except for

Architecture	W/A	Hard Quant	PQN
MobileNetV2	4/4 _{MP}	69.71	69.25
	3/3 _{MP}	65.65	64.21
MobileNetV3-Small	4/4 _{MP}	64.23	53.92
	3/3 _{MP}	57.14	14.61

Table 3: Training with hard quantization and STE compared versus PQN during the MPQ phase of QAT. Validation accuracy (%) on ImageNet.

Arch.	W/A	0%	25%	50%	75%	100%
MNv2	4/4 _{MP}	68.63	69.55	69.71	69.7	67.81
	3/3 _{MP}	62.71	65.83	65.65	65.56	62.47
MNv3	4/4 _{MP}	59.62	64.39	64.23	64.34	62.66
	3/3 _{MP}	43.70	57.49	57.14	56.9	54.94

Table 4: MobileNetV2 and MobileNetV3-Small for various fractions (%) of the total training time. 0% corresponds to using QBitOpt on a subset of training data before training. Validation accuracy (%) on ImageNet.

MobileNetV2 with a 4 bits target where greedy integers perform better. In figure 4 (in Appendix D.3), we illustrate the progression of the bitwidth allocation during training. Moreover, we observe that for greedy integer, some bitwidths oscillate between two values which could introduce noise in the network optimization. On the other hand, fractional bitwidths progress much smoother during *mixed precision* phase, however at the end of the phase, there is a jump once the bitwidth gets rounded to a viable natural number for the *fine-tuning* phase. For the remainder of the experiments, we use the greedy integer method.

Pseudo-noise vs. hard quantization In the *mixed precision* phase, we jointly train the network weights leveraging QAT while updating the bitwidth allocation. Since updating the sensitivities is independent of the QAT update, we can choose any quantization method for QAT. Two natural choices are pseudo-quantization noise (PQN), which we used to derive our sensitivity metric, and hard-quantization with STE, which is often used in most QAT literature [24, 26, 14, 30]. In table 3, we compare both choices. Hard quantization with STE outperforms PQN for all models tested. Particularly for MobileNetV3-Small, the performance gap between the methods is substantial. We hypothesize that it is important for the network to early in training adapt to the exact quantization noise present during inference.

Two-phase mixed-precision training In this section, we study the impact of our two-phased mixed-precision training. In table 4, we show the results with various divisions of the training time between the *mixed precision* phase and the *fine-tuning* phase. Most importantly, these results demonstrate that both phases are important for obtaining the best mixed-precision performance. When only fine-tuning is used after a post-training bitwidth allocation (0% column), the performance significantly degrades compared to updating the bitwidth during training using QBitOpt — especially for 3-bit quantization. This shows that our novel QBitOps algorithm, which integrates bitwidth allocation into QAT, compares favorably to common post-training MPQ followed by fine-tuning. On the other hand, updating the bitwidth allocations up to the conclusion of training (100% column) adversely affects performance. The non-constant bitwidths add extra noise to the optimization procedure. This can be harmful in the last phase of training, as discussed in the previous paragraph, and is a common issue known in the MPQ literature. As such, most training-based approaches circumvent this by not updating the bitwidths towards the end of training [41, 38].

While having both phases is clearly essential in finding the best mixed-precision network, our framework is relatively stable to the exact division of the training time between them. For most networks and bitwidth, using between 25% and 75% for the mixed precision phase leads to good results.

5.3. Comparison to existing methods

We compare QbitOpt to other QAT approaches: (a) LSQ [14] for fixed-precision quantization; (b) *differentiable quantization* (DQ) [40] and *noise injection pseudo quantization* (NIPQ) [38] for mixed-precision. For a fair comparison, we have re-implemented DQ and NIPQ, adhering to their implementation details and modifying them appropriately to match our constraints and quantization assumptions. We performed an extensive hyperparameter search on the regularization strength λ and learning rates. For further details, refer to appendix D.

In table 5, we present the results for MobileNetV2, MobileNetV3-Small and EfficientNet-Lite on the ImageNet classification benchmark. QbitOpt significantly outperforms the strong fixed-precision LSQ baseline in all cases. Especially for the challenging to quantize MobileNetV3-Small, the accuracy improves up to 5%. We also outperform *differential quantization* (DQ), in terms of quantized accuracy, despite DQ achieving average bitwidths higher than the specified target. These results showcase one of the biggest strengths of our method, namely that the resource constraint is met exactly. This contrasts with the other mixed-precision methods, where we struggle to meet the target constraint even after an extensive hyperparameter

	Method	W/A	Avg. Bits	Acc. (%)
MobileNetV2	Full-precision	32/32	-	71.72
	LSQ [14]	4/4	4.0	69.45 ^{0.07}
	DQ [40]	4/4 _{MP}	4.08 ^{0.01}	68.55 ^{0.30}
	QBitOpt (ours)	4/4 _{MP}	4.0	69.75 ^{0.04}
	LSQ [14]	3/3	3.0	65.17 ^{0.12}
	DQ [40]	3/3 _{MP}	3.10 ^{0.01}	64.86 ^{0.07}
QBitOpt (ours)	3/3 _{MP}	3.0	65.71 ^{0.07}	
MobileNetV3-Small	Full-precision	32/32	-	67.67
	LSQ [14]	4/4	4.0	61.58 ^{0.07}
	DQ [40]	4/4 _{MP}	4.10 ^{0.01}	63.60 ^{0.02}
	QBitOpt (ours)	4/4 _{MP}	4.0	64.34 ^{0.06}
	LSQ [14]	3/3	3.0	51.92 ^{0.04}
	DQ [40]	3/3 _{MP}	3.09 ^{0.01}	57.26 ^{0.44}
QBitOpt (ours)	3/3 _{MP}	3.0	57.36 ^{0.17}	
EfficientNet-lite	Full-precision	32/32	-	75.42
	LSQ [14]	4/4	4.0	72.93 ^{0.09}
	DQ [40]	4/4 _{MP}	4.08 ^{0.01}	72.29 ^{0.21}
	QBitOpt (ours)	4/4 _{MP}	4.0	73.32 ^{0.10}
	LSQ [14]	3/3	3.0	69.62 ^{0.15}
	DQ [40]	3/3 _{MP}	3.13 ^{0.01}	68.82 ^{0.21}
QBitOpt (ours)	3/3 _{MP}	3.0	70.00 ^{0.07}	

Table 5: ImageNet image classification benchmark. We report validation accuracy, target bitwidth (W/A) and the achieved average bitwidth (Avg. Bits) averaged over 3 seeds (STD in superscript). † denotes only 2 seeds due to unstable training.

search.

In table 6, we compare QBitOpt to NIPQ [38] using the same constraint as in their work (cf. equation (14)): the *per-element average bitwidth* constraint. NIPQ outperforms QBitOpt in terms of accuracy by a small margin ($\leq 0.6\%$) for MobileNetV2, but NIPQ struggles to reach to specified bitwidth constraint for activations in the case of 4 bits or weights in the case of 3 bits. The accuracy gap grows further to 2.44% in the case of the MobileNetV3-Small W4A4, but so does the gap of NIPQ from the target bitwidth. Despite aggressive regularization and extensive search, we could not bring the activations’ average bitwidth under 4.25 bits. For MobileNetV3-Small W3A3, NIPQ closes the gap to target bitwidth for activations but is still not on target. While NIPQ is indeed a strong mixed-precision benchmark, this study once again highlights the main advantage of our QBitOpt, which is ease and speed of experimentation. In the case of QBitOpt, we only had to search for learning rate schedules to achieve quantized models that are guaranteed

	Method	Bits(W/A)	Acc. (%)
MNv2	NIPQ [38]	3.90/4.18	71.00
	QBitOpt (ours)	4/4	70.39 ^{0.05}
	NIPQ [38]	3.18/3.06	68.76
	QBitOpt (ours)	3/3	68.44 ^{0.0}
MNv3-Small	NIPQ [38]	3.94/4.25	64.89
	QBitOpt (ours)	4/4	62.45 ^{0.21}
	NIPQ [38]	2.98/3.09	59.54
	QBitOpt (ours)	3/3	57.82 ^{0.29}

Table 6: Average bitwidth weighed by tensor size constraint independently optimized for weights and activation. We report validation accuracy and the achieved *average bitwidth* for weights/activations over 3 seeds (STD in superscript) for QBitOpt and results for a single seed for NIPQ.

to satisfy the constraint, while for NIPQ, we required extensive and time-consuming search across learning rates and regularization strength.

6. Conclusion

In this work, we introduced QBitOpt, a novel algorithm for allocating bitwidths under strict resource constraints. QBitOpt leverages techniques from convex optimization and QAT to obtain high-performing neural networks in the sense of resource requirements and task performance without the need to carefully balance task loss and resource cost through a cumbersome hyper-parameter. We justify using Hessian-based sensitivities, traditionally only used for fully converged neural networks, during training. Our greedy integer method and efficient FIT approximation allow us to regularly update the bitwidth allocations during quantization-aware training.

We demonstrate the efficacy of QBitOpt on several architectures in which QBitOpt compares favorably to competing fixed-precision and mixed-precision approaches. We further examine various properties of our method through ablation studies. Specifically, we show that regularly reallocating bitwidths during training is crucial for optimal performance. We identify improved sensitivities that more closely capture quantization perturbation and the design of convex relaxations for hardware constraints as exciting avenues for future research.

References

- [1] Milad Alizadeh, Arash Behboodi, Mart van Baalen, Christos Louizos, Tijmen Blankevoort, and Max Welling. Gradient l-1 regularization for quantization robustness. *arXiv preprint arXiv:2002.07520*, 2020. 4

- [2] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018. 5
- [3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2
- [4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 4
- [5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020. 1
- [6] 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. 2
- [7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014. 5
- [8] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. Differentiable model compression via pseudo quantization noise. Apr. 2021. 3, 5
- [9] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016. 4
- [10] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020. 5
- [11] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 4, 5
- [12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1
- [13] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. Differentiable model compression via pseudo quantization noise. *arXiv preprint arXiv:2104.09987*, 2021. 2
- [14] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020. 1, 2, 5, 6, 7, 8
- [15] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 1
- [16] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022. 1
- [17] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021. 2, 5
- [18] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998. 2
- [19] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015. 5
- [20] Hai Victor Habi, Roy H Jennings, and Arnon Netzer. Hmq: Hardware friendly mixed precision quantization block for cnns. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16*, pages 448–463. Springer, 2020. 5
- [21] Geoffrey Hinton. Neural networks for machine learning, lectures 15b. 2012. 2, 5
- [22] M. Horowitz. 1.1 computing’s energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014. 1
- [23] B. Ham J. Lee, D. Kim. Network quantization with element-wise gradient scaling. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 2
- [24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 1, 5, 7
- [25] Sambhav R. Jain, Albert Gural, Michael Wu, and Chris Dick. Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware. *arxiv preprint arxiv:1903.08066*, 2019. 1, 2
- [26] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 1, 7
- [27] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. {BRECQ}: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021. 1
- [28] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, 2020. 1
- [29] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort.

- A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021. 1, 2, 5, 6
- [30] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. Overcoming oscillations in quantization-aware training. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16318–16330. PMLR, 17–23 Jul 2022. 1, 2, 6, 7, 11
- [31] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *International Conference on Computer Vision (ICCV)*, 2019. 1
- [32] Nilesh Prasad Pandey, Markus Nagel, Mart van Baalen, Yin Huang, Chirag Patel, and Tijmen Blankevoort. A practical mixed precision algorithm for post-training quantization. *arXiv preprint arXiv:2302.05397*, 2023. 5
- [33] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981. 5
- [34] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018. 1
- [35] Manuele Rusci, Marco Fariselli, Alessandro Capotondi, and Luca Benini. Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers. In *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning: Second International Workshop, IoT Streams 2020, and First International Workshop, ITEM 2020, Co-located with ECML/PKDD 2020, Ghent, Belgium, September 14-18, 2020, Revised Selected Papers 2*, pages 296–308. Springer, 2020. 5
- [36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. 6
- [37] Clemens JS Schaefer, Elfie Guo, Caitlin Stanton, Xiaofan Zhang, Tom Jablin, Navid Lambert-Shirzad, Jian Li, Chichen Chou, Yu Emma Wang, and Siddharth Joshi. Mixed precision post training quantization of neural networks with sensitivity guided search. *arXiv preprint arXiv:2302.01382*, 2023. 6
- [38] Juncheol Shin, Junhyuk So, Sein Park, Seungyeop Kang, Sungjoo Yoo, and Eunhyeok Park. Nipq: Noise proxy-based integrated pseudo-quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3852–3861, June 2023. 3, 5, 6, 7, 8
- [39] Hugo Touvron, Matthieu Cord, and Hervé Jégou. Deit iii: Revenge of the vit. In *Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*, pages 516–533. Springer, 2022. 1
- [40] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. In *International Conference on Learning Representations (ICLR)*, 2020. 1, 2, 3, 5, 7, 8, 12, 13
- [41] Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5741–5752. Curran Associates, Inc., 2020. 3, 5, 6, 7
- [42] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 5
- [43] Peisong Wang, Chen Qiang, He Xiangyu, and Jian Cheng. Towards accurate post-training network quantization via bit-split and stitching. In *Proceedings of the 37nd International Conference on Machine Learning (ICML)*, pages 243–252, July 2020. 1
- [44] Bernard Widrow. A study of rough amplitude quantization by means of nyquist sampling theory. *IRE Transactions on Circuit Theory*, 3(4):266–276, 1956. 2
- [45] Bernard Widrow. Statistical analysis of amplitude-quantized sampled-data systems. *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, 79(6):555–568, 1961. 2
- [46] B Widrow and M Liu. Statistical theory of quantization. *IEEE Trans. Instrum. Meas.*, 45(2):353–361, Feb. 1996. 2
- [47] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999. 3
- [48] 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. 5
- [49] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, and Kurt Keutzer. Hawq-v3: Dyadic neural network quantization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11875–11886. PMLR, 18–24 Jul 2021. 5
- [50] Ben Zandonati, Adrian Alan Pol, Maurizio Pierini, Olya Sirkin, and Tal Kopetz. Fit: A metric for model sensitivity. *arXiv preprint arXiv:2210.08502*, 2022. 4, 5
- [51] Sijie Zhao, Tao Yue, and Xuemei Hu. Distribution-aware adaptive multi-bit quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9281–9290, 2021. 5