A. Activation quantization

In section 3.3, we formulated the QBitOpt optimization objective in terms of network parameters $\boldsymbol{\Theta}$. However, we commonly also quantize the activations in neural network quantization. Fortunately, our method extends easily to activations too, as it is trivial to compute quantization sensitivities for activations. As a reminder, in section 3.1, we defined the Hessian diagonal $\boldsymbol{h}$ as an example of quantization sensitivity. By computing the *hessian sensitivities* with respect to activations, we can infer bitwidth allocation for activation as per equation 9.

Considering an arbitrary neural network F with L layers as a composition of functions:

$$F(\boldsymbol{x}) = \left[f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1\right](\boldsymbol{x}), \tag{16}$$

we can compute the sensitivity of the i^{th} activation $\boldsymbol{z}_i = \left[f_i \circ \cdots \circ f_1\right](\boldsymbol{x})$ by simply considering the sub-network:

$$F_{i+1} = \left[f_L \circ f_{L-1} \circ \cdots \circ f_{i+1}\right] \tag{17}$$

The *hessian sensitivity* is now given by:

$$\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{\partial^2}{\partial \boldsymbol{z}_i^2}\mathcal{L}\left(F_{i+1}(\boldsymbol{z})\right)\Big|_{\boldsymbol{z}=\boldsymbol{z}_i}\right] \tag{18}$$

The sensitivity can be computed over the full dataset $\mathcal{D}$ using an exponential moving average, as described in section 3.1.1. In practice, one may consider applying separate constraints to the parameter and activation quantizer bitwidth. We leave this investigation for future work.

B. Inner bitwidth minimization

For convenience, we restate (7) here:

$$\min_{\boldsymbol{\Theta}} \min_{\boldsymbol{b}} \mathbb{E}_{\boldsymbol{\epsilon}}\left[\mathcal{L}\left(\boldsymbol{\Theta} + \frac{\boldsymbol{\alpha}}{2^{\boldsymbol{b}}-1}\boldsymbol{\epsilon}\right)\right] \quad \text{s.t. } \widehat{\pi}(\boldsymbol{b}) \preceq 0. \tag{19}$$

As presented in the main text(cf. section 3.1), we aim to solve an approximation to the inner minimization. The solution to this is then used to take a gradient step for the outer minimization. First, let $\boldsymbol{\delta} = \boldsymbol{\alpha}/(2^{\boldsymbol{b}}-1)$. Then we approximate the objective using a second-order Taylor approximation:

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\epsilon}}\left[\mathcal{L}(\boldsymbol{\Theta}+\boldsymbol{\delta\epsilon})\right]\\ &\approx \mathbb{E}_{\boldsymbol{\epsilon}}\left[\mathcal{L}(\boldsymbol{\Theta}) + (\boldsymbol{\delta\epsilon})^\top \nabla_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}) + \frac{1}{2}(\boldsymbol{\delta\epsilon})^\top \nabla^2_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta})(\boldsymbol{\delta\epsilon})\right]\end{aligned} \tag{20}$$

The first term is constant with respect to $\boldsymbol{b}$ and the second term equals zero as $\mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}] = 0$. This leaves the following optimization objective:

$$\begin{aligned}&\frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[(\boldsymbol{\delta\epsilon})^\top\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right](\boldsymbol{\delta\epsilon})\right]\\ &= \frac{1}{2}\mathrm{Tr}\left\{\mathbb{E}_{\boldsymbol{\epsilon}}[(\boldsymbol{\delta\epsilon})^\top\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right](\boldsymbol{\delta\epsilon})]\right\}\\ &\overset{(a)}{=} \frac{1}{2}\mathrm{Tr}\left\{\mathbb{E}_{\boldsymbol{\epsilon}}[(\boldsymbol{\delta\epsilon})(\boldsymbol{\delta\epsilon})^\top]\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right]\right\}\\ &= \frac{1}{2}\mathrm{Tr}\left\{\mathrm{diag}(\boldsymbol{\delta})\mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon\epsilon}^\top]\,\mathrm{diag}(\boldsymbol{\delta})\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right]\right\}\\ &\overset{(b)}{=} \frac{1}{24}\mathrm{Tr}\left\{\mathrm{diag}(\boldsymbol{\delta})\mathrm{diag}(\boldsymbol{\delta})\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right]\right\}\\ &= \frac{1}{24}\mathrm{Tr}\left\{\mathrm{diag}(\boldsymbol{\delta})^2\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right]\right\}\\ &= \frac{1}{24}\sum_i^{|\boldsymbol{\Theta}|}\left[\nabla^2\mathcal{L}(\boldsymbol{\Theta})\right]_{ii}\boldsymbol{\delta}_{ii}^2.\end{aligned} \tag{21}$$

Here, (a) uses $\mathrm{Tr}(A^\top B) = \mathrm{Tr}(BA^\top)$ where A and B are two $m \times n$ real matrices and (b) follows from $\mathbb{E}[\boldsymbol{\epsilon\epsilon}^\top] = \boldsymbol{I}/12$. Finally, dropping the multiplicative constants that do not affect the minimization problem and re-substituting $\boldsymbol{\delta}$, we obtain our optimization problem:

$$\begin{aligned}\boldsymbol{b}^* = \min_{\boldsymbol{b}} \boldsymbol{h}^\top\left(\frac{\boldsymbol{\alpha}}{2^{\boldsymbol{b}}-1}\right)^2, \quad \boldsymbol{h}_i = \nabla^2\mathcal{L}(\boldsymbol{\Theta})_{ii}\\ \text{subject to } \widehat{\pi}(\boldsymbol{b}) \preceq 0.\end{aligned} \tag{22}$$

C. Experimental configuration

C.1. Optimization

All experiments are trained on NVIDIA GPUs using Python 3.8.10, PyTorch v1.11, and Torchvision v0.12. We found that using separate optimizers for the model (SGD) and quantization parameters (Adam) leads to stabler training and higher accuracy across architectures and methods. The optimizer and learning rate schedules for both optimizers can be found in table 7. *Phase-1* refers to QAT with bitwidth reallocation using QBitOpt, whereas in *phase-2*, we fix the quantizers' bitwidth and fine-tune with QAT. At the end of each training epoch, we re-estimate the batch-normalization statistics using 50 batches of training data, as per [30].

C.2. QBitOpt configuration

During *phase-1* of training, we calculate an exponential-moving average of each quantizer's FIT sensitivity S_q with a momentum of 0.9, as shown in algorithm 1. Because sensitivities change quite smoothly during training (see fig. 2), we decided to compute them only every two training iterations to reduce training time. We infer a new bitwidth allocation by solving the optimization every $\tau = 250$ training iterations. We found that QBitOpt is not very sensitive to this hyperparameter, and decreasing τ did not lead to better

Model	W/A	Epochs		SGD - Model parameters				Adam - Quant. parameters	
		Phase-1	Phase-2	LR	η_{min}	Warmup	WD	LR	WD
MobileNetV2	4/4	15	15	0.0033	3.3×10^{-6}	-	1.0×10^{-5}	1.0×10^{-5}	0.0
	3/3	15	15	0.01	1.0×10^{-5}	-	1.0×10^{-5}	1.0×10^{-5}	0.0
EfficientNet-Lite	4/4	15	15	0.0033	3.3×10^{-6}	-	1.0×10^{-5}	1.0×10^{-5}	0.0
	3/3	15	15	0.01	1.0×10^{-5}	-	1.0×10^{-5}	1.0×10^{-5}	0.0
MobileNetV3-Small	4/4	20	20	0.07	7.0×10^{-5}	4	1.0×10^{-5}	1.0×10^{-5}	0.0
	3/3	20	20	0.05	5.0×10^{-5}	4	0.0	1.0×10^{-5}	0.0

Table 7: Optimization configuration: all model parameters (except for the quantization parameters) are optimized using SGD with a momentum of 0.9 and a cosine annealing schedule for learning rate. We use a separate Adam optimizer with a constant learning rate for the quantization parameters. WD: weight decay; Warmup: number of epoch for linear learning rate warmup; η_{min}: final learning rate of cosine decay.

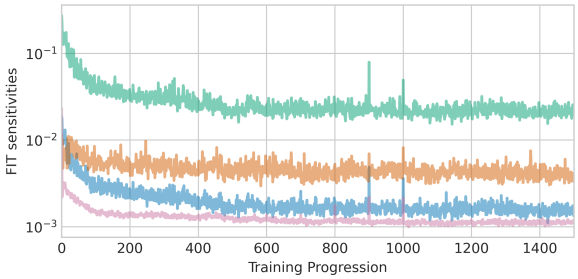


Figure 2: FIT sensitivities progression over time (y-axis in logarithmic scale)

results. In fact, given that we learn the quantization range α using gradients, τ should be large enough to allow the quantization range to adapt to the latest bitwidth allocation.

C.3. Model checkpoints

- MobileNetV2: https://github.com/tonylins/pytorch-mobilenet-v2
- MobileNetV3-Small: https://pytorch.org/vision/0.12/models
- EfficientNet-Lite: https://github.com/huggingface/pytorch-image-models

D. Additional experimental results

D.1. Differential quantization (DQ)

In tables 9, 8 & 10, we present the results of our hyperparameter search over the regularization strength λ in the objective of *differential quantization* (DQ) [40]. In contrast to QBitOpt, the method fails to reach the target average bitwidth exactly. Notably, for MobileNetV3-Small, we have to extend the grid search significantly to achieve a satisfactory average bitwidth solution. This study demonstrates the power of QBitOpt that does not really on a scalarized multi-objective loss. Instead, QBitOpt guarantees the exact constraint, enabling the user to focus on tuning the QAT hyperparameters and achieve the highest task performance under the specified constraint. Please note that the results in bold are shown in the final results table 5.

Method	Avg. bits	Acc. (%)
DQ [$\lambda = 0.3$]	4.299	64.38
DQ [$\lambda = 0.5$]	4.187	63.87
DQ [$\lambda = 0.7$]	4.150	63.92
DQ [$\lambda = 1.0$]	**4.093**	**63.61**
DQ [$\lambda = 1.4$]	4.065	63.56
DQ [$\lambda = 2.0$]	4.065	63.42
DQ [$\lambda = 3.0$]	4.037	63.44
DQ [$\lambda = 5.0$]	4.028	63.44
DQ [$\lambda = 8.0$]	4.009	63.27
QBitOpt	**4.0**	**64.23**
DQ [$\lambda = 0.3$]	3.542	60.45
DQ [$\lambda = 0.5$]	3.402	59.62
DQ [$\lambda = 0.7$]	3.290	58.48
DQ [$\lambda = 1.0$]	**3.215**	**58.33**
DQ [$\lambda = 1.4$]	3.159	57.95
DQ [$\lambda = 2.0$]	3.121	57.63
DQ [$\lambda = 3.0$]	3.084	56.86
DQ [$\lambda = 5.0$]	3.056	55.70
DQ [$\lambda = 8.0$]	3.037	56.89
QBitOpt	**3.0**	**57.14**

Table 8: MobileNetV3-Small on ImaneNet. DQ [40] results for different regularization strength λ, including our QBitOpt result. The values in bold are used in table 5.

Method	Avg. bits	Acc. (%)
DQ [$\lambda = 0.3$]	4.210	69.09
DQ [$\lambda = 0.5$]	4.143	68.59
DQ [$\lambda = 0.7$]	4.105	68.57
DQ [$\lambda = 1.0$]	**4.076**	**68.50**
DQ [$\lambda = 1.4$]	4.067	68.51
DQ [$\lambda = 2.0$]	4.038	68.80
QBitOpt	**4.0**	**69.71**
DQ [$\lambda = 0.3$]	3.286	66.33
DQ [$\lambda = 0.5$]	3.229	65.44
DQ [$\lambda = 0.7$]	3.124	64.78
DQ [$\lambda = 1.0$]	**3.114**	**64.94**
DQ [$\lambda = 1.4$]	3.086	64.27
DQ [$\lambda = 2.0$]	3.048	64.18
QBitOpt	**3.0**	**65.65**

Table 9: MobileNetV2 on ImaneNet. DQ [40] results for different regularization strength λ, including our QBitOpt result. The values in bold are used in table 5.

Method	Avg. bits	Acc. (%)
DQ [$\lambda = 0.3$]	4.242	72.48
DQ [$\lambda = 0.5$]	4.141	72.61
DQ [$\lambda = 0.7$]	4.111	72.50
DQ [$\lambda = 1.0$]	**4.081**	**72.14**
DQ [$\lambda = 1.4$]	4.061	72.16
DQ [$\lambda = 2.0$]	4.051	72.20
QBitOpt	**4.0**	**73.43**
DQ [$\lambda = 0.3$]	3.354	70.03
DQ [$\lambda = 0.5$]	3.212	69.24
DQ [$\lambda = 0.7$]	3.141	68.95
DQ [$\lambda = 1.0$]	**3.121**	**68.86**
DQ [$\lambda = 1.4$]	3.101	68.48
DQ [$\lambda = 2.0$]	3.051	68.33
QBitOpt	**3.0**	**70.04**

Table 10: EfficientNet-Lite on ImaneNet. DQ [40] results for different regularization strength λ, including our QBitOpt result. The values in bold are used in table 5.

D.2. Effect of α-factor in exponential moving average on sensitivity estimation

In Section 3.1.1, it was mentioned that an exponential moving average (EMA) is used to estimate sensitivities over multiple batches during training. Specifically, given the observed sensitivity e_t at timestep t and $\alpha \in (0, 1]$, the EMA $\mathcal{E}_t$ is defined as:

$$\mathcal{E}_t = (1 - \alpha)\mathcal{E}_{t-1} + \alpha e_t, \quad \mathcal{E}_1 = e_1. \tag{23}$$

The choice of α determines how much weight is put on past observations compared to new observations. E.g., when $\alpha = 1$, only the latest observations are used. Figure 3 shows results for different architectures trained using various values for $\alpha \in \{0.05, 0.25, 0.5, 0.75, 0.9, 1.0\}$. This shows that the choice of α is insignificant, and one could choose not to include the exponential moving average estimator. This result agrees with the progression of FIT estimates shown in Figure 2, where we showed that sensitivities are relatively stable during training. However, since this figure also shows several outliers, we use the EMA estimator for the results presented in the main text.

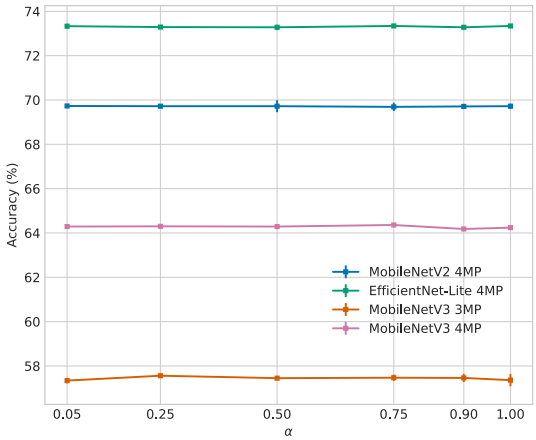


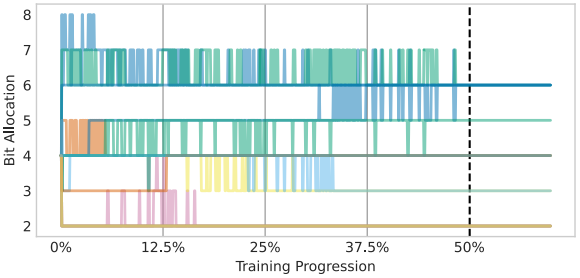
Figure 3: Accuracy obtained for different settings of α for the Exponential Moving Average sensitivity estimation. The whiskers, if visible, show the standard deviation over three seeds.

D.3. Bitwidth allocation during training

When using QBitOpt, parameters and activation bitwidths/quantization levels change during training. Figure 4 shows the bitwidth allocation over the course of training for a random subset of layers.

D.4. QBitOpt bitwidth allocation

In figure 5, we illustrate the final bitwidth allocation resulting from QBitOpt on MobileNetV2 and MobileNetV3-Small with a 3-bit average target bitwidth. In both cases, QBitOpt allocated more bits in the first and last layers of the networks, which is consistent with empirical observations from existing literature. In addition, in MobileNetV2, we also observe that QbitOpt tends to assign higher bitwidth to activations compared to weights.



(a) Bitwidth allocation using *greedy integer* bit optimization

(b) Bitwidth allocation using fractional bit optimization

Figure 4: Bitwidth allocation during training

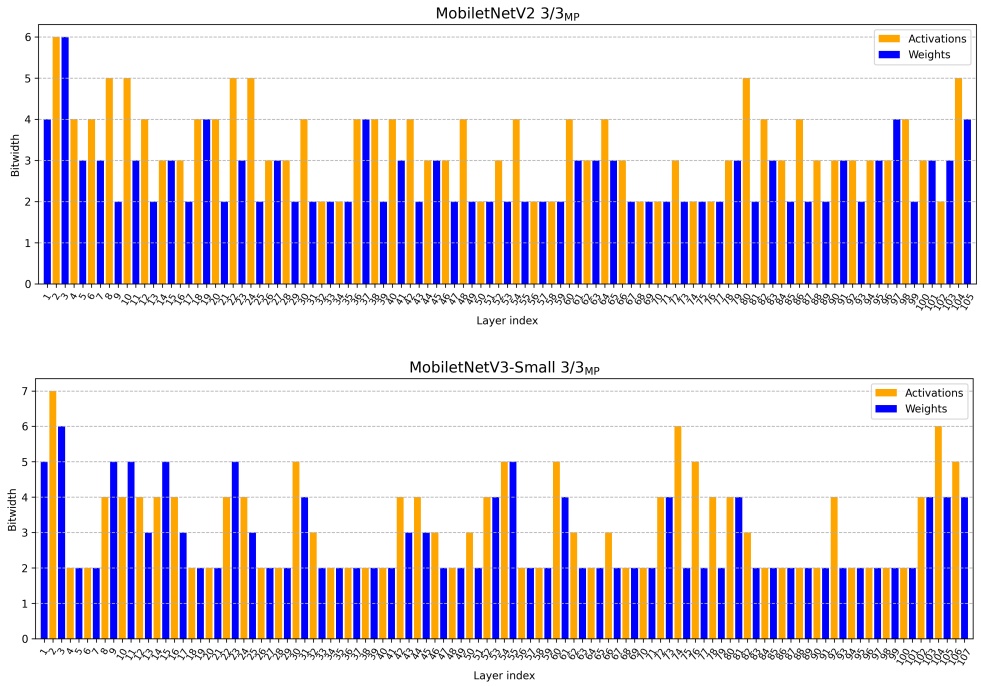


Figure 5: Final bitwidth allocation for MobileNetV2 and MobileNetV3-Small using QBitOpt with 3-bit average target bitwidth $3/3_{\text{MP}}$.