A Details in experiment

A.1 Implementation

We use Pytorch (Paszke et al., 2019) for all the implementation. For experiments on CIFAR-10 and CIFAR-100, we use a single NVidia A6000 GPU. Training of 200 epochs took a few hours. For experiments on ImageNet, we use A100 DGX with four GPUs. Training of 100 epochs took five days.

A.2 Network architecture

CIFAR-10/CIFAR-100 We use ResNet18 (He et al., 2016a) for this experiment. To adapt to the smaller input size of 32 (compared with 224 of ImageNet (Deng et al., 2009)), we follow the protocol of CIFAR-10 example used in Lighting Bolts library (Falcon & Cho, 2020); replace the first 7×7 convolution layer (stride 2) for 3×3 convolution layer (stride 1) and remove the following `max-pooling` layer. The output dimensions of the final linear layer are changed according to the dataset's number of classes (10 for CIFAR-10 and 100 for CIFAR-100).

ImageNet We use ConvNeXt-B (Liu et al., 2022). To utilize the activation sparsity, we replace `GELU` (Hendrycks & Gimpel, 2016) activation with `ReLU` activation. From their intensive ablation study (figure 2 in their paper), this change of activation function has very little impact on accuracy.

A.3 Data Augmentation

CIFAR-10/CIFAR-100 We follow the convention in the literature for training the datasets; randomly crop of 32×32 region after 4 pixel padding then applies random horizontal flip.

ImageNet We use the data augmentation techniques used in ConvNeXt (Liu et al., 2022) (settings for the fine-tuning stage) except for the stochastic depth. Specifically, we use random erasing of the probability 0.25 and label smoothing of factor 0.1. We use PyTorch Image Models library (timm) (Wightman, 2019) for the data augmentation.

A.4 Detail in Weight-Pruning

For Weight-Pruning, we use magnitude-based pruning using *Hoyer-Square* Yang et al. (2020) for the magnitude computation combined with the LSQ-based quantization. In Weight-Pruning, weights elements that are zero after the quantization is pruned. The algorithm for the Weight-Pruning is similar to the one proposed in the *neural-wiring* Wortsman et al. (2019), *edge-popup* Ramanujan et al. (2020), and the ones adopted by *Hoyer-Square* Yang et al. (2020). We use quantization by LSQ instead of thresholding, and the elements of the weight that are zero after the quantization are pruned. Therefore, pruned weight can be active again, which is key to finding a good pruned network having good accuracy/energy tradeoff.

We adopt the Weight-Pruning setting for a fair comparison with the proposed Bit-Pruning. In Bit-Pruning, quantized weight is converted into binary format, and elements that become zeros are pruned. In other words, both methods prune a value smaller than the threshold (e.g., half the quantization scale) where the difference between the target weight ($\mathbf{0}$ for Weight-Pruning and $\hat{\boldsymbol{W}}$ for Bit-Pruning) is evaluated using Hoyer-Square.

A.5 Detail in Bit-Pruning

A.5.1 Balancing parameter $\lambda^{(l)}$ for proximal weight

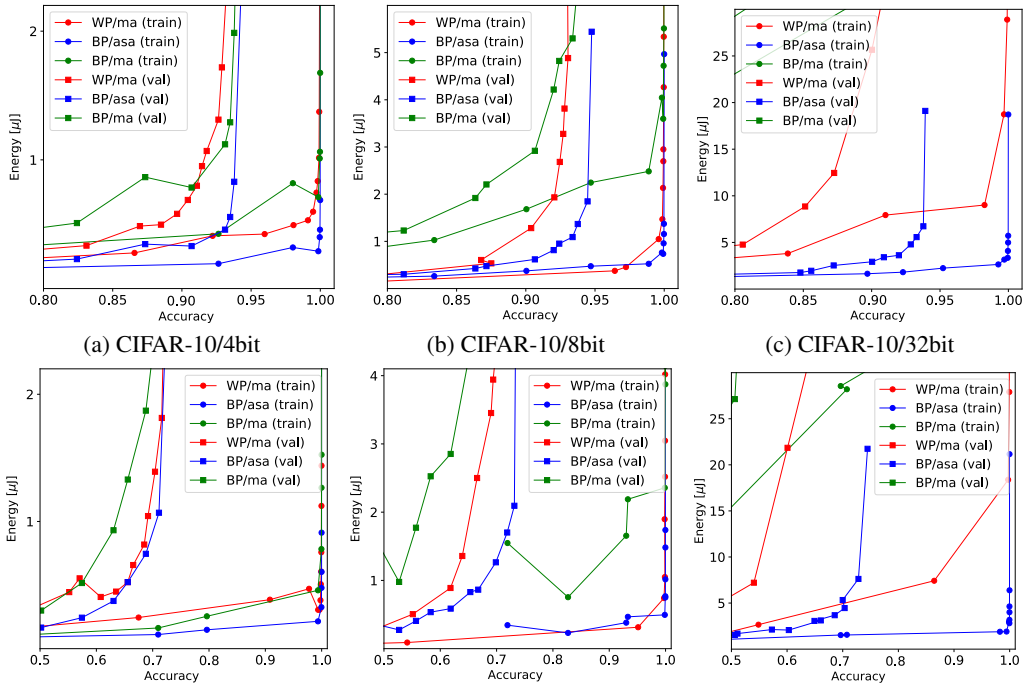
As a preliminary study investigating the effectiveness of Bit-Pruning, we use the same λ for entire layers. But we can use a different value of λ for each layer. It can be set proportional to the number of operations for the given layers, or it could take the susceptibility of the weight change to the output layer into account. Learning the layer-wise λ end-to-end may also be possible using task loss. However, this requires an additional ingenuity to approximate the gradient w.r.t. λ (because

of the non-differentiability f $\arg\min$ operation). In any case, it will introduce an additional hyper-parameter to be tuned. In this study, we focus on comparing the Weight-Pruning and Bit-Pruning in a simple and fair setup as possible. Therefore we left the exploration in this direction for future work.

A.5.2 SCHEDULING THE STRENGTH η OF SPARSITY REGULARIZATION

We use constant η across the entire training sequence both for Weight-Pruning and Bit-Pruning. We adopted this setting because we want to compare the Weight-Pruning and Bit-Pruning in a simple and fair configuration as possible.

Scheduling of the η will improve the accuracy/energy tradeoff; however, we prioritize the comparison on simple and fair configuration, and we left the exploration of the dynamic scheduling of η as future work.



(a) CIFAR-10/4bit (b) CIFAR-10/8bit (c) CIFAR-10/32bit

(d) CIFAR-100/4bit (e) CIFAR-100/8bit (f) CIFAR-100/32bit

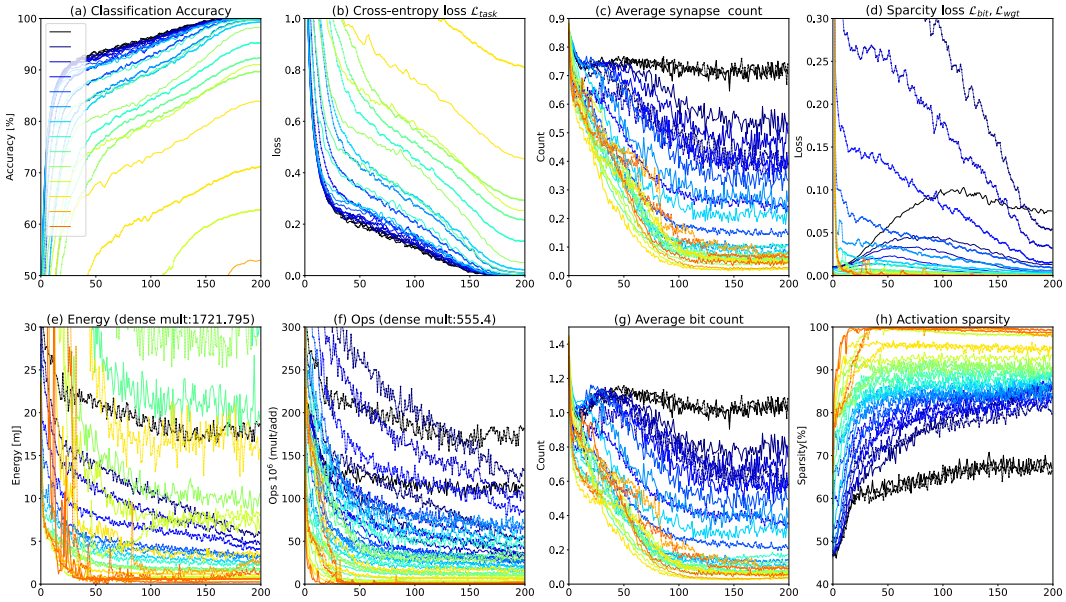
Figure 8: Accuracy/energy tradeoff of ResNet18 trained on CIFAR-10 and CIFAR-100 for difference activation quantization level. Bit-Pruning (`add-shift-add` mode) (BP/asa) and Weight-Pruning (`mult-add` mode) (WP/ma) are compared. Bit-Pruning always uses an 8bit basis, and Weight-Pruning uses the same bit for weight as activation. The result of Bit-Pruning in `mult-add` mode (BP/ma) is provided for reference.

B ADDITIONAL EXPERIMENT RESULT

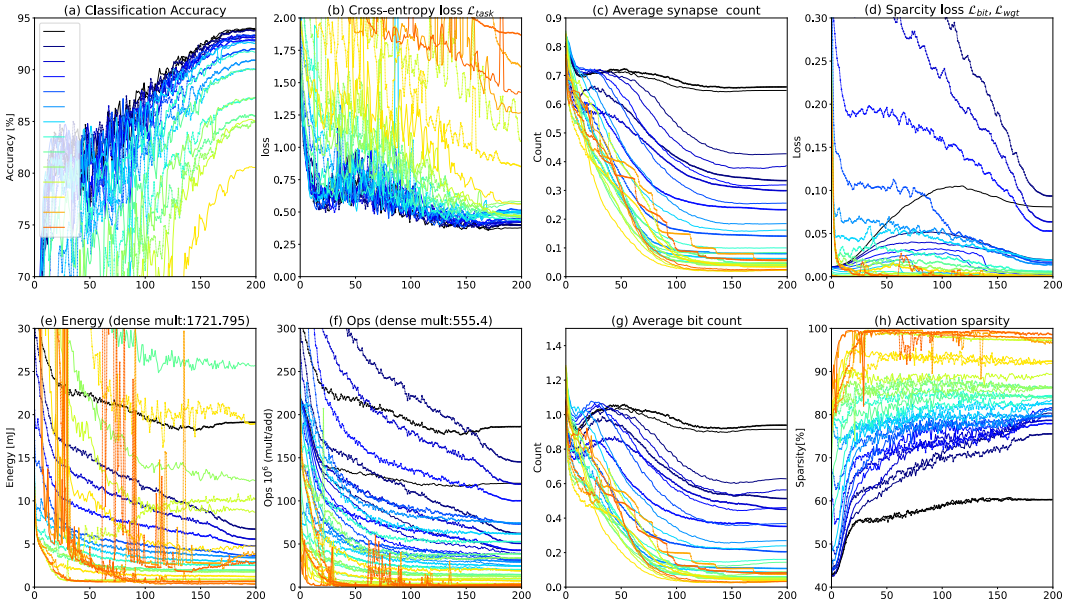
As a supplemental result of the accuracy/energy comparison experiment (Figure 4) presented in the main paper, the results, including the training accuracy, are shown in Figure 8 for CIFAR-10 (Figure 8a, 8b, 8c) and CIFAR-100 (Figure 8d, 8e, 8f). The training loss of Bit-Pruning is lower than that of Weight-Pruning for the same computational energy. It suggests that the `add-shift-add` network has more capacity than the `mult-add` network given the same energy.

Figure 9 - 10 shows the evolution of major statics during training epochs. The results presented in Figure 8 in the main paper is a summary of (a) and (e) of Figure 9 - 10.

Using a large weight for the sparsity regularization η (both for Weight-Pruning and Bit-Pruning), both weight and activation become sparse (Figure 9 - 10 (c), (g), (h)). This is because the sparsity regularization ($\mathcal{L}_{wgt}$ of (3) for Weight-Pruning and $\mathcal{L}_{bit}$ of (7) for Bit-Pruning) can be reduced not only by sparsifying the `mult` or `add` but also by suppressing the activation by promoting the activation to the minus region of `ReLU` activation function.



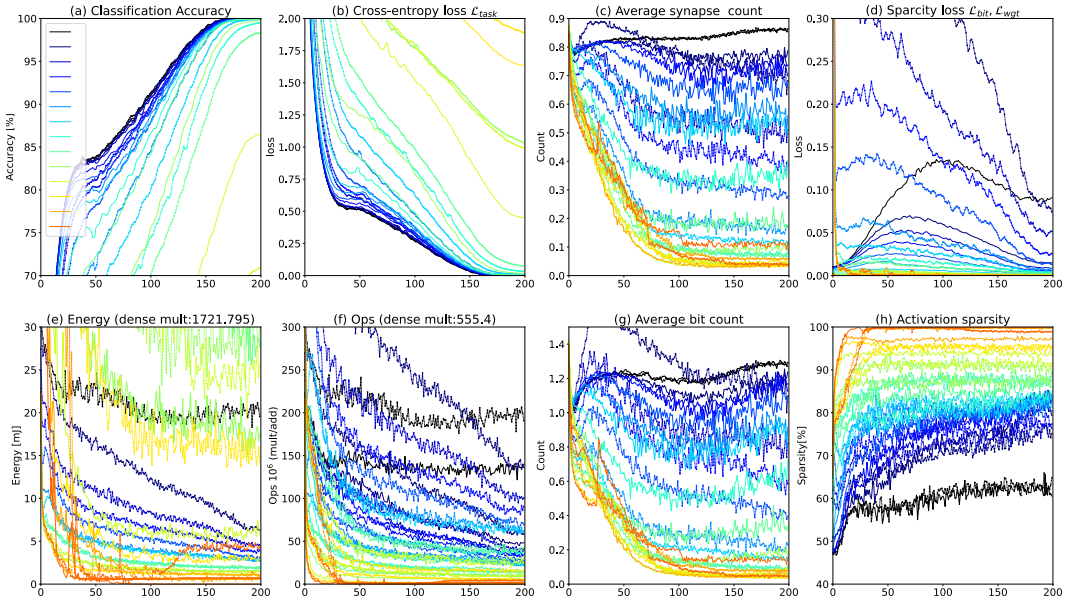
(a) Training



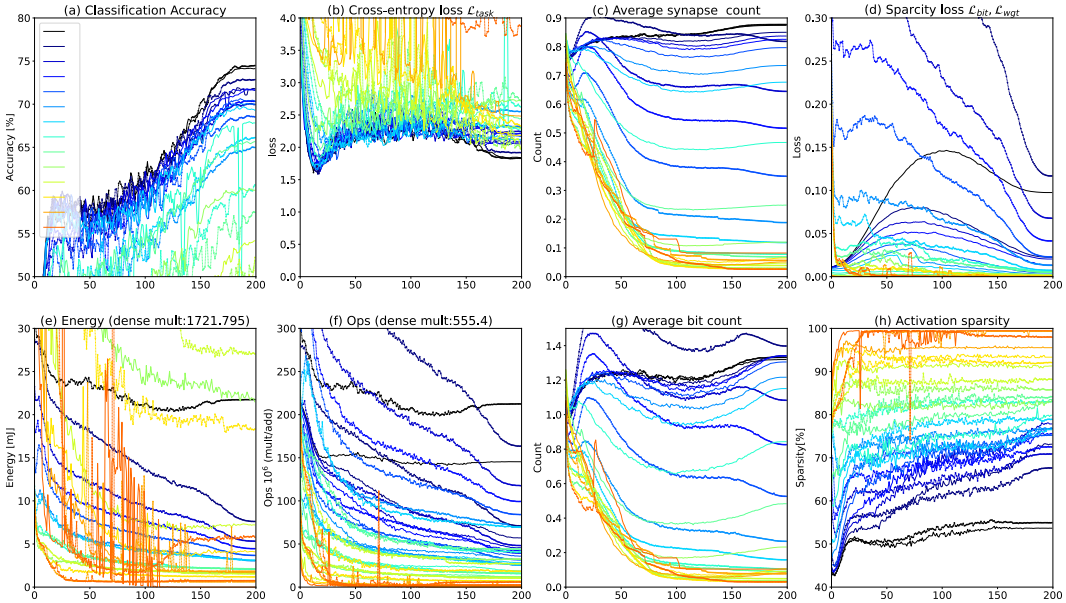
(b) Validation

Figure 9: Evolution of major statics on CIFAR-10 (Krizhevsky et al., a). Weight-Pruning (solid line) vs. Bit-Pruning (dash line with small dot). ResNet18 (He et al., 2016a). Color encode the strengths of sparsity regularization η ($\mathcal{L}_{wgt}$ for Bit-Pruning and $\mathcal{L}_{bit}$ for Bit-Pruning). Black: η is zero, Blue: η is small, Orange: η is large. (a) Classification accuracy in [%], (b) Task loss $\mathcal{L}_{task}$ (cross-entropy loss). (c) Average nonzero synapse count with respect to the original synapse connection (i.e., 1.0 is equivalent to dense connection.) (d) Loss for sparsification (`mult` for Wight-Pruning ($\mathcal{L}_{wgt}$) and `add` for Bit-Pruning ($\mathcal{L}_{bit}$)). (e) Estimated energy consumption from sparse `mult-add` and `add-shift-add` based on the statistics in Table 1. (f) Operation count of `mult` (Weight-Pruning) and `add` (Bit-Pruning). (g) Average bit count for each neuron (initial bit count is about 4 when initialized by Kaiming uniform for 8-bit weight as shown in figure 6a). (h) Activation sparsity (i.e., the ratio of zero elements in $\boldsymbol{X}$)

.



(a) Training



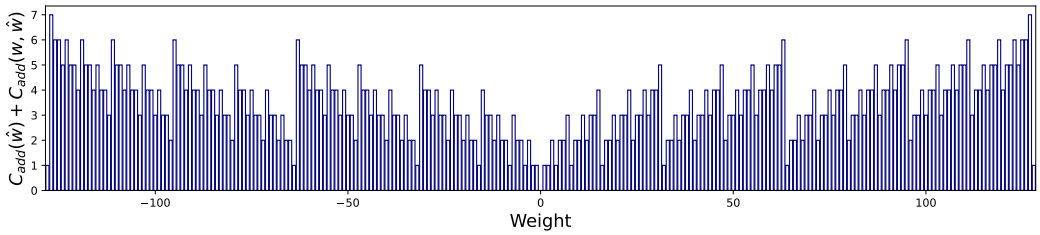
(b) Validation

Figure 10: Evolution of major statics on CIFAR-100 (Krizhevsky et al., b). Weight-Pruning (solid line) vs. Bit-Pruning (dash line with small dot). ResNet18 (He et al., 2016a). Color encode the strengths of sparsity regularization η ($\mathcal{L}_{wgt}$ for Bit-Pruning and $\mathcal{L}_{bit}$ for Bit-Pruning). Black: η is zero, Blue: η is small, Orange: η is large. (a) Classification accuracy in [%], (b) Task loss $\mathcal{L}_{task}$ (cross-entropy loss). (c) Average nonzero synapse count with respect to the original synapse connection (i.e., 1.0 is equivalent to dense connection.) (d) Loss for sparsification (`mult` for Wight-Pruning ($\mathcal{L}_{wgt}$) and `add` for Bit-Pruning ($\mathcal{L}_{bit}$)). (e) Estimated energy consumption from sparse `mult-add` and `add-shift-add` based on the statistics in Table 1. (f) Operation count of `mult` (Weight-Pruning) and `add` (Bit-Pruning). (g) Average bit count for each neuron (initial bit count is about 4 when initialized by Kaiming uniform for 8bit weight as shown in figure 6a). (h) Activation sparsity (i.e., ratio of zero element in $\boldsymbol{X}$)

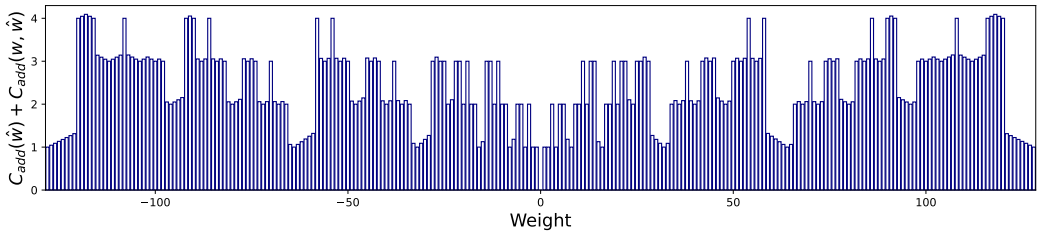
.

C LOSS LANDSCAPE OF PROXIMAL WEIGHT

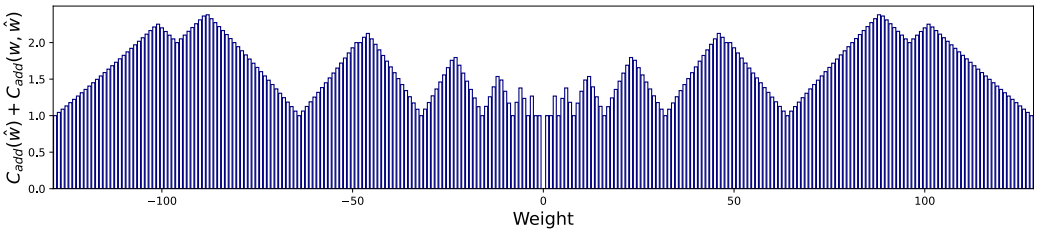
The loss landscape of the bit sparsity regularization of (7) is visualized in Figure 11. There will be multiple local minima, depending on the magnitude of $\lambda^{(l)}$. When $\lambda^{(l)}$ is very large, the local minima will be PoT and zero; on the other hand, when $\lambda^{(l)}$ is zero, it coincides with the `add` cost $\mathcal{C}_{add}$ of Figure 3a. Depending on the current weight value, the weights are guided toward the local minima except in the case when $\lambda^{(l)} = 0.0$ (In this case $\hat{\mathbf{W}}$=$\mathbf{W}$ and we won't have any gradient from the bit-sparsity regularization of (7)).



(a) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 0.0$)



(b) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 0.1$)



(c) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 1.0$)

Figure 11: Loss landscape of *bit-sparsity regularization* $\mathcal{L}_{bit} := \mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$. (b) and (c) corresponds to Figure 3b and Figure 3c in the main paper.

D MORE DISCUSSION ON COMPUTATIONAL EFFICIENCY

As discussed in the main paper (Section 3.3), we roughly get $E_{\texttt{mult}} \approx M E_{\texttt{add}}$ from Table 1 in the case of ASIC; therefore, the computational efficiency of Bit-Pruning over Weight-Pruning is almost determined by the ratio of obtained connection density $\sigma_{\texttt{add}}/\sigma_{\texttt{mult}}$.

When the network weight is initialized with a uniform random variable, then about 50% of the bits in $\mathcal{B}$ is one (Figure 6a); in this case, their computational efficiency is comparable (same order).

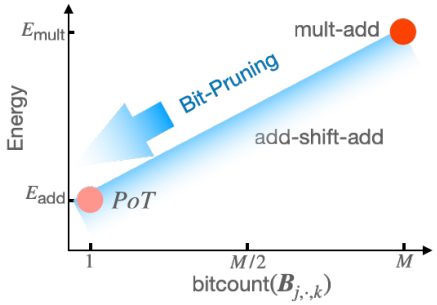


Figure 12: Fine-grained pruning of Bit-Pruning. The energy cost for the nonzero weight of `add-shift-add` ranges from $E_{\texttt{add}}$ to $E_{\texttt{mult}} \approx M E_{\texttt{add}}$. Bit-Pruning aims for $E_{\texttt{add}}$, by sparsifying $\mathcal{B}$.

By the fine-grained pruning, we expect that the average $\sigma_{\texttt{add}}$ in a Bit-Pruned network is much smaller than the average $\sigma_{\texttt{mult}}$ in a Weight-Pruned network when both networks achieve comparable accuracy (Figure 12). In the extreme case where all the nonzero weights in a network are represented as PoT, the computational cost in `add-shift-add` representation is M times smaller than that in `mult-add` representation. The `mult`-equivalent computation can be executed using roughly $E_{\texttt{add}}$ instead of $E_{\texttt{mult}}$.

E ABLATION

In this section, we compare Bit-Pruning and Weight-Pruning in a wide range of configurations which we could not cover in the main paper.

The additional results are shown only for CIFAR10 using 8-bit activation.

E.1 SMALLER SIZED NETWORK

Wight-Pruning and Bit-Pruning assume some redundancy in weights to be pruned. To investigate the behavior of both methods in a situation where the number of possible unnecessary weights is scarce, we've conducted experiments using the narrow version of ResNet18 (narrow ResNet18). The narrow ResNet18 has $8\times$ less input and output channels w.r.t the original ResNet18 we've used in the main experiments, except the input and output layer. In this situation, we can not expect significant pruning, either on weights or bits.

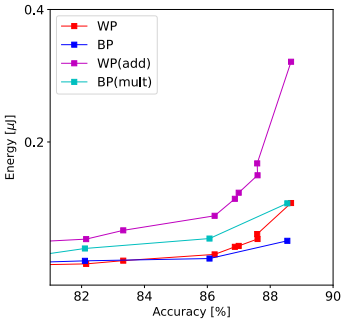


Figure 13: Ablation result from using narrow ($8\times$) network. Results using ResNet18 model with 8-bit activation on CIFAR10.

Figure 13 shows the result from using the narrow ResNet18. As expected, even when there is no sparsity regularization, e.g., $\eta = 0$, we already observe significant degeneration in accuracy compared with the original full-sized ResNet18, both on Weight-Pruning and Bit-Pruning. Increasing the weight for the regularization η, we observed the reduction in energy at the cost of accuracy degeneration in both methods. However, we observed a similar trend as in the case of the full-size network; the Bit-Pruning shows a better tradeoff than the Weight-Pruning.

E.2 DIFFERENT BALANCING PARAMETER $\lambda^{(l)}$ FOR THE PROXIMAL WEIGHT

The proximal weight $\hat{\mathcal{W}}$ is computed by finding the *best* balance between the gain from reducing the bit cost $\mathcal{C}_{bit}$ and proximity from the current weight. The proximity is evaluated by the weight moving cost function $\mathcal{C}_{mv}$. Intuitively, $\mathcal{C}_{add}$ encourages $\hat{\boldsymbol{W}}_{j,k}$ to be sparse in the binary format while $\mathcal{C}_{mv}$ keeps it close to the current weight $\boldsymbol{W}_{j,k}$. The *best* balance is determined by choice of λ and $\mathcal{C}_{mv}$. The proximal weight $\hat{\mathcal{W}}$ will be sparser in binary representation when λ is large.

In the main paper, we choose to use $\lambda^{(l)} = 1.0$ for all layers (Tab.3). We choose this value without careful parameter search (e.g., by evaluating the test accuracy) but by simply looking at the loss landscape of Figure 11. We consider it will induce sparsity as its local minimum is mainly composed of one nonzero element (except two nonzeroes at 96 and all zero at 0). To see the results when $\lambda^{(l)}$ is small where there are more local minimums (Figure 11-(b)), we conducted an experiment using a smaller value of $\lambda^{(l)} = 0.1$ for all layers.

Figure 14 shows the result from using smaller value of $\lambda^{(l)}$ ($\lambda^{(l)} = 0.1$). We observe the opposite results from the main experiment; the accuracy/energy tradeoff of Bit-Pruning is worse than that of Weight-Pruning.

The degenerated tradeoff may be attributed to the too-conservative proximal weight, or it may come from the difficulty in training using the loss function that has too many local minima (Figure 11-(b)).

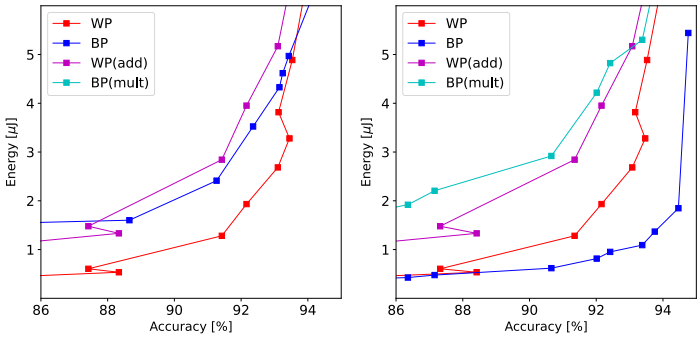


Figure 14: Ablation result from using smaller λ ($\lambda = 0.1$, left) compared with the results using larger λ ($\lambda = 1.0$) (right, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

E.3 Fine tuning vs. Training from scratch

In the main paper, we trained the ResNet18 network from scratch for CIFAR10/100 and fine-tuned the ConvNeXt network from a pre-trained network for ImageNet. We trained the network from scratch because we wanted to compare Weight-Pruning and Bit-Pruning in a fair and straightforward setting as possible. We were concerned the results may be affected by the pre-trained network. (We use a pre-trained network for ConvNeXt because we could not run large-scale training using of larger dataset such as ImageNet-22K.)

Nonetheless, it is noteworthy to see which performs better to consider the application scenario of Bit-Pruning (and also Weight-Pruning). For this, we conducted experiments to compare the fine-tuning and the train-from-scratch. For the pre-trained network, we used the network trained as $\eta = 0$ for both Weight-Pruning and Bit-Pruning. We employed the training procedure used in train-from-scratch.

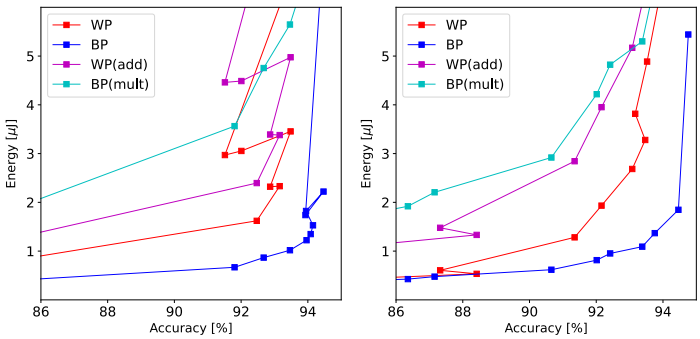


Figure 15: Ablation result of fine-tuning using pre-trained network (left) compared with the results by training from scratch (right, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

Figure 15 compares the results from fine-tuning and train-from-scratch (same results from the Figure 4-(b) in the main paper). The results are almost the same on the high-accuracy regime, but we observe faster convergence in the case of fine-tuning. We observe the instability in the low energy regime (large η). This instability might be attributed to the learning-rate scheduler (e.g., warm-up, etc.) designed for the train-from-scratch scenario.

Bit-Pruning (and Weight-Pruning) could be used for both scenarios (fine-tuning and train-from-scratch). But given the similar results (at least on the high-accuracy regime) and faster convergence, it may be beneficial to use a pre-trained network when available.

E.4 UNIFORM WEIGHT MOVING COST $\mathcal{C}_{mv}$ WITH $p = 1.0$

The proximal weight $\hat{\mathcal{W}}$ is computed by finding the *best* balance between the gain from reducing the bit cost $\mathcal{C}_{bit}$ and proximity from the current weight evaluated by the weight moving cost function $\mathcal{C}_{mv}$. Intuitively, $\mathcal{C}_{add}$ encourages $\hat{\boldsymbol{W}}_{j,k}$ to be sparse in the binary format while $\mathcal{C}_{mv}$ keeps it close to the current weight $\boldsymbol{W}_{j,k}$. The *best* balance is determined by choice of λ and $\mathcal{C}_{mv}$. In section E.2, we ablate the balancing parameter $\lambda^{(l)}$; here, we ablate the choice of the weight moving cost C_{mv}.

In the main experiments, we use $p = 1/2$ because we consider the changes in weight value when its norm is small to have more effect on the accuracy than when it is large. For example, changes of 2 when the weight was 1 (200% change) may have a more significant impact on accuracy than the weight was 200 (1%). When $p = 1/2$, C_{mv} changes a lot when its norm is small and does not change much when its magnitude is significant.

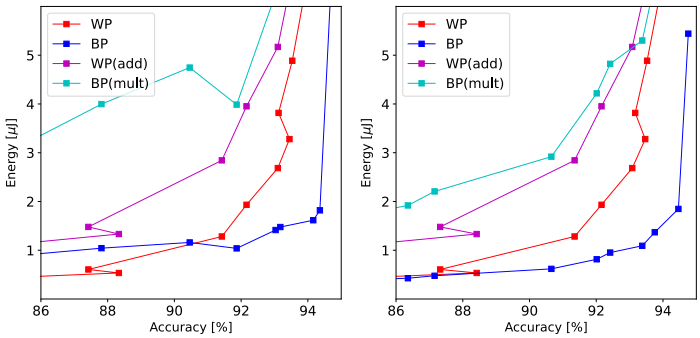
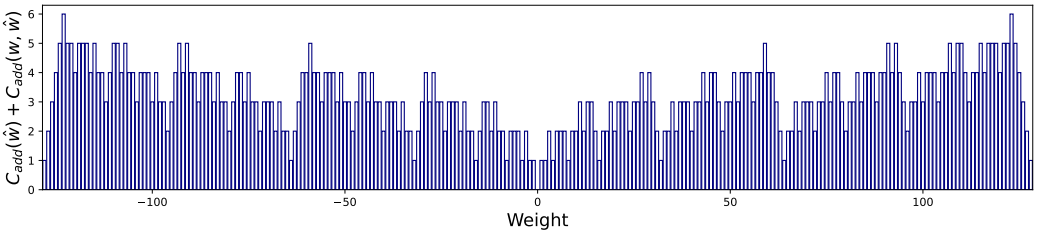


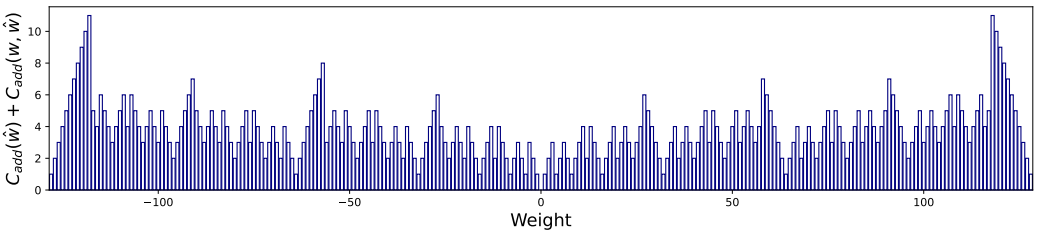
Figure 16: Ablation result of using uniform weight moving cost (left, $\mathcal{C}_{mv}$ with $p = 1.0$) compared with norm dependant weight moving cost (right, $\mathcal{C}_{mv}$ with $p = 1/2$, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

We conducted the ablation experiments using the moving cost C_{mv} with $p = 1.0$; this choice assumes the change in weight will equally affect the accuracy regardless of the norm of the current weight. Figure 17 visualizes the loss landscape of the bit-sparsity regularization when $p = 1.0$ which is significantly different from the case when $p = 1/2$ shown in Figure 11.

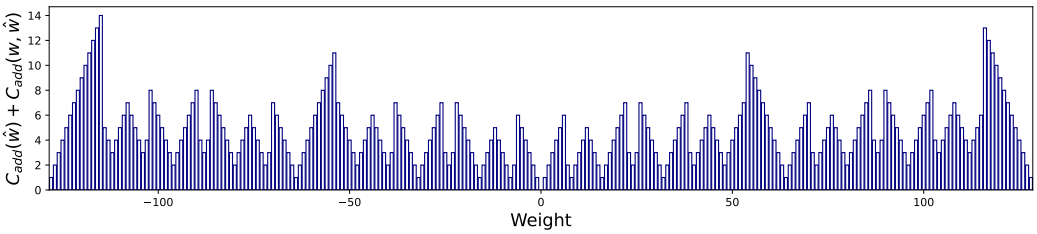
Figure 16 shows the results when $p = 1.0$. We observe a degradation in the accuracy/energy tradeoff. We suspect this is because the $\mathcal{L}_{bit}$ with $p = 1.0$ does not reflect the tradeoff of changing the value of w to the accuracy. This result suggests that we could expect a better tradeoff by using *smarter* C_{mv}, e.g., by learning from data (as we discussed in future work in Sec.6).



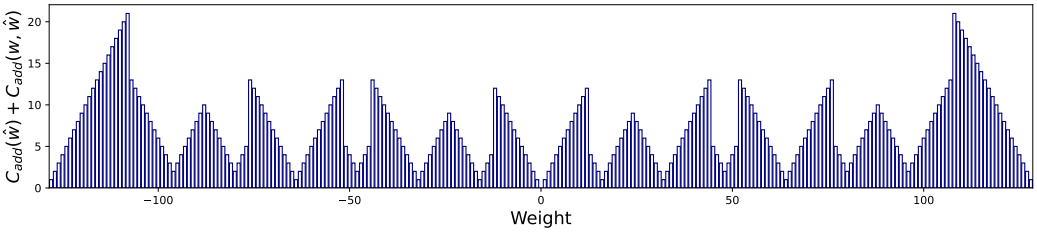
(a) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 1.0$)



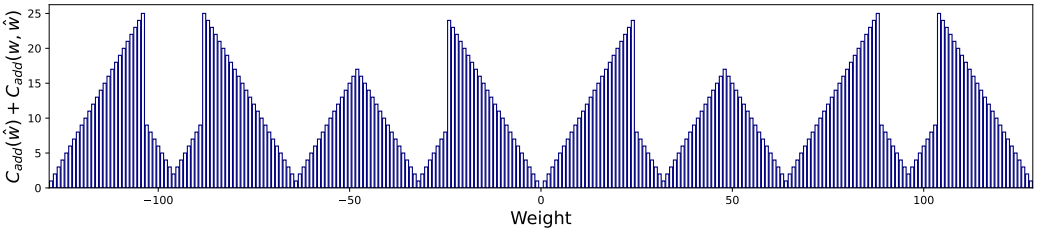
(b) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 2.0$)



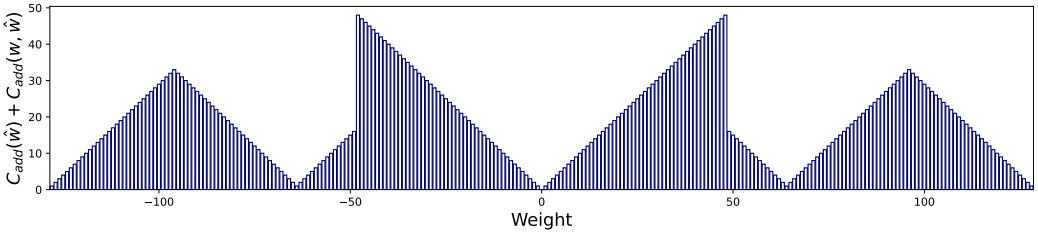
(c) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 4.0$)



(d) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 8.0$)



(e) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 16.0$)



(f) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 32.0$)

Figure 17: Loss landscape of *bit-sparsity regularization* $\mathcal{L}_{bit} := \mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$ for $\mathcal{C}_{mv}$ with $p = 1.0$

F COMPARISON WITH THE BIT-SLICE LOSS

Inducing group-wise bit sparsity (bit-slice sparsity) is explored to realize efficient DNN for emerging resistive random-access memory (ReRAM)-based DNN accelerators Zhang et al. (2019). In the ReRAM-based accelerators, each crossbar can not hold the whole 8-bit weight; therefore, a single weight element is distributed into the different crossbars, each having a 2-bit which they call a bit-sliced representation. In this hardware restriction, their goal is to sparsify each slice. The sparsity within each slice is induced by minimizing the bit-slice l_1 loss, defined as a l_1 norm of each sliced weight value (in their case, each slice has 2bit). By reducing the l_1 norm for each slice, the weight as a whole is also expected to have fewer nonzero elements in binary representation.

Ours and Zhang et al. (2019) are different in motivation; therefore, their technique for inducing bit-sparsity is also different. Their goal is to induce the sparsity within the slice; on the hand, our goal is to induce the sparsity in `add` of the proposed `add-shift-add` formulations. They induce the sparsity by minimizing the l_1 loss within each slice and do not consider the continuity of weight as a whole; on the hand, we consider the continuity of the whole weight when inducing the sparsity to keep the change of weight value minimum to prevent the accuracy deterioration by the significant change in the weight value.

Although the motivation is different, Bit-Pruning and bit-slice sparsification of Zhang et al. (2019) share some similarities in the scene in that both aim at realizing the sparse representation of weight in binary format. Therefore, we conducted additional experiments by comparing them in terms of their ability to induce sparsity.

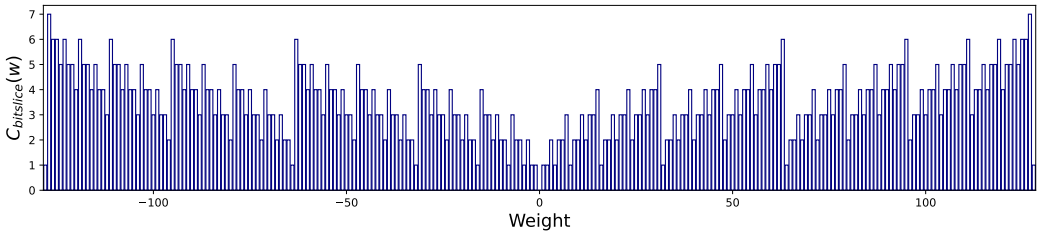
Figure 18 visualize the loss landscape of their bit-slice regularization $\mathcal{L}_{bitslice}$. Because the l_1 loss is computed inside the slice, it can not take the state of other bits into account; therefore, it can not take the continuity of the weight into account. When the number of slices is 1 (Figure 18-(d)), the bit-slice l_1 loss reduces to the $l1$ regularization for the weight value; therefore, we can not expect the sparsity as it simply reduces the norm of the weight. When the number of slices equals the bit width M (Figure 18-(a)), it guides each bit to zero, ignoring the weight value's continuity. When the number of slices is 2 (Figure 18-(c)), which is their experimental setup), it only considers the continuity in the slice and ignores the continuity as weight as a whole.

For example, consider the case when $w = 63$ (6 nonzero bit), then our proximal weight is $w = 64$ (1 nonzero bit). Both weights are guided toward the proximal value, which differs only 1 in the weight value. Still, nonzero bits are reduced significantly (from 6 to 1). On the other hand in case of the bit-slice l_1 loss it guides the weight towards $63_{10} = (00|11|11|11)_2 \mapsto 42 = (00|10|10|10)$. The corresponding weight value changes drastically; therefore, we'll expect a significant drop in accuracy. Worse still, the gain for the bit sparsity is small (from 6 to 3). Our bit-sparsity loss guides the weight that maximizes the gain while preventing the weight from changing a lot by simultaneously considering the proximity of the weight value and the number of bits.

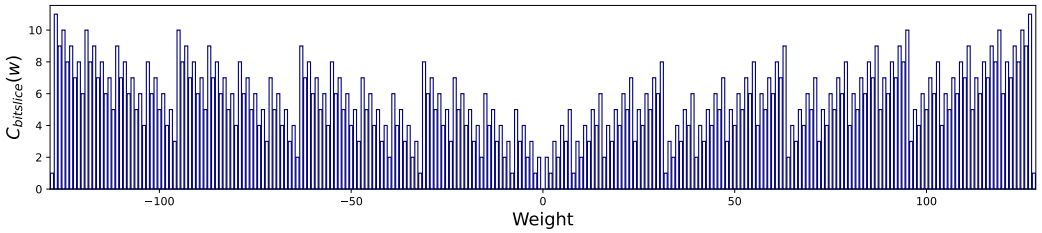
We conducted experiments to compare the result of our bit-sparsity regularization and their bit-slice l_1 regulation. We modified their loss function `calc_l1_and_zero_ratio` from their provided code[8] to be used in our Bit-Pruning framework. We also use l_1 loss (instead of Hoyer-Square loss) for our bit-sparsity regularization for a fair comparison.

Figure 19 compare their performance. Our bit-sparsity regularization (l_1 version) shows a better tradeoff than the bit-slice l_1 regulation.

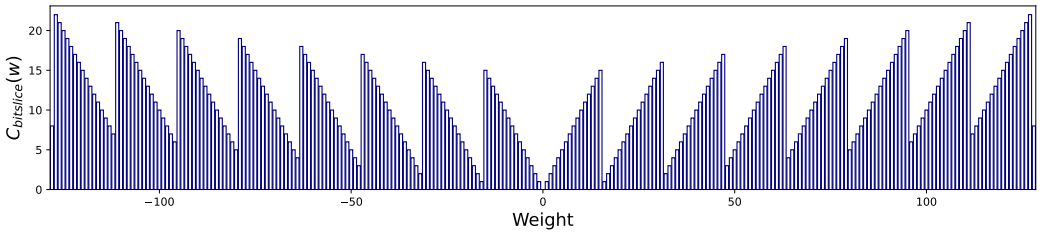
[8] http://github.com/zjysteven/bitslice_sparsity



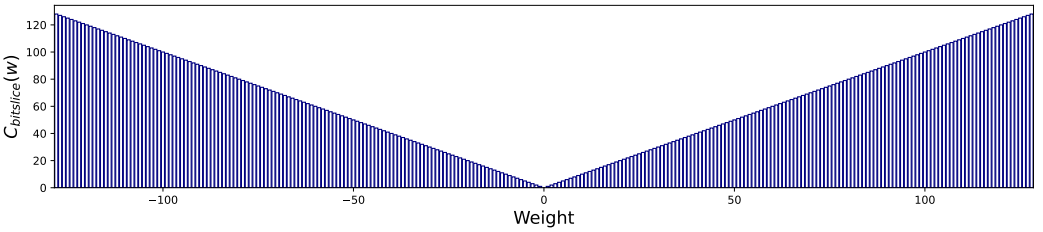
(a) Loss landscape of bit-slice l_1 regularization $\mathcal{L}_{bitslice}$ (slice=8)



(b) Loss landscape of bit-slice l_1 regularization $\mathcal{L}_{bitslice}$ (slice=4), same setting as Zhang et al. (2019)



(c) Loss landscape of bit-slice l_1 regularization $\mathcal{L}_{bitslice}$ (slice=2)



(d) Loss landscape of bit-slice l_1 regularization $\mathcal{L}_{bitslice}$ (slice=1)

Figure 18: Loss landscape of *bit-slice regularization* $\mathcal{L}_{bitslice}$ Zhang et al. (2019). When the number of slices is 1, it reduces to the $l1$ regularization. When the number of slices is a number of bit M, it simply guides each bit to zero; this is equivalent to $\lambda = 0$ in our bit-slice l_1 regularization.

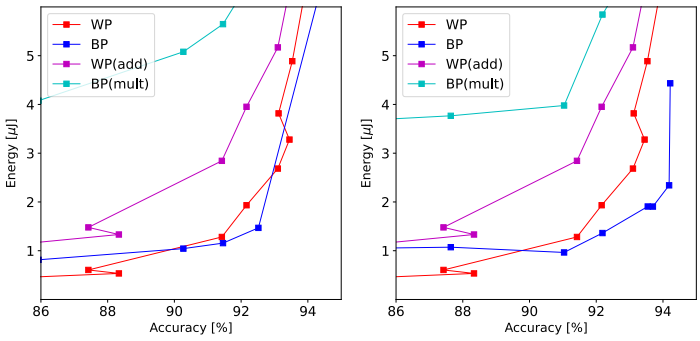


Figure 19: Ablation result using bit-slice l_1 regularization (left) compared with our bit-sparsity regularization (right, using l_1 loss instead of square-Hoyer). Results using ResNet18 model with 8-bit activation on CIFAR10.

G WEIGHTED BIT-PRUNING REGULARIZATION

Our Bit-Pruning loss is defined as the difference (in Hoyer-Square sense) between the proximal weight $\hat{\boldsymbol{W}}$ and current weight $\boldsymbol{W}$. The proximal weight takes both bit-cost and proximity to the current weight into account (8). It is a compromised point (found by solving the optimization of (8)) that reduces the bit-cost while keeping the change from the current weight small as possible. The bit-cost C_{bit} of the computed proximal weight is always less than or equal to the current weight regardless of the choice of the moving cost $\mathcal{C}_{mv}$; therefore, sparsity always induces. However, the bit-sparsity regularization does NOT consider the possible reduction of the bit-cost C_{bit} once the proximal weight is determined.

Let's consider a case when $w = 63$ and $w = 65$. They would have the same proximal $\hat{w} = 64$. Obviously, $w = 63$ has more nonzero bits than $w = 65$. Using the bit-sparsity regularization of (7), both $w = 63$ and $w = 65$ are guided toward $w = 64$; therefore, both weights are directed toward the sparse in binary representation having only one nonzero bit. But we'll expect more gain when $w = 63$ becomes $w = 64$ ($6 \mapsto 1$) than $w = 65$ becomes $w = 64$ ($2 \mapsto 1$). The bit-sparsity regularization of (7) does not consider this for computing the gradient. Instead, they are treated equally, ignoring their different gain in bit-cost. To incorporate these gains into account, we consider using weight to reflect the gain; the weighted version of the bit sparsity regularization is defined as follows:

$$\mathcal{L}_{bit}(\mathcal{W}) = \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot (\boldsymbol{W}^{(l)} - \hat{\boldsymbol{W}}^{(l)}) \right|_0,$$

$$\mathcal{L}_{bit\_weighted}(\mathcal{W}) = \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot (\boldsymbol{W}^{(l)} - \hat{\boldsymbol{W}}^{(l)})(C_{add}(\boldsymbol{W}^{(l)}) - C_{add}(\hat{\boldsymbol{W}}^{(l)})) \right|_0, \tag{10}$$

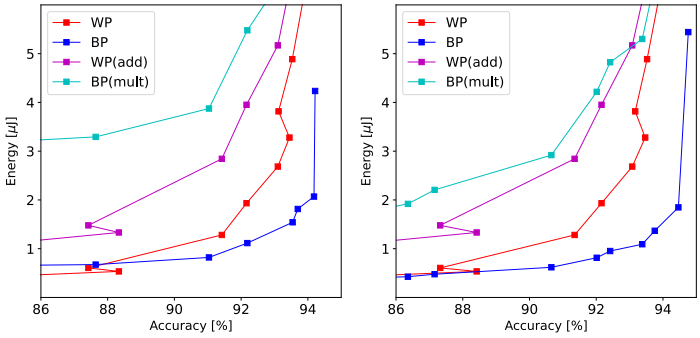


Figure 20: Ablation result of using weighted bit sparsify regularization of (10) (left) compared with the normal bit sparsify regularization of (7). Results using ResNet18 model with 8-bit activation on CIFAR10.

Figure 20 shows the results. Opposed to our expectation, we could not observe a noticeable improvement in the accuracy-energy tradeoff. We have not found the reason yet; we may observe some improvement using a different optimizer, model designs, or datasets. In this work, we focused on comparing Bit-Pruning and Weight-Pruning in a simple and fair setup; therefore, we left the explanation of more advanced Bit-Pruning regularization for future work.

H ADDITIONAL DISCUSSION WITH RELATED WORKS

H.1 LOW-RANK FACTORIZATION

Low-rank factorization of matrix multiplication is often utilized to reduce the number of $\texttt{mult}$ (Howard et al., 2019). The factorization can be realized as depthwise or 1x1 convolution; they are utilized in a computationally efficient network such as MobileNet (Howard et al., 2017) or ConvNeXt (which we evaluate in our experiment). Recent research demonstrates that the low-rank form could be learned by optimization (Idelbayev & Carreira-Perpinán, 2020). Our approach is orthogonal to this approach, and it can be combined, c.f., applying Bit-Pruning for the learnable low-rank network to reduce the computation cost further.

H.2 BSQ

BSQ Yang et al. (2021) proposed a novel method for realizing gradient-based optimization of the layer-wise precision, targeting the hardware supporting the mixed precision. They formulate the optimization of layer-wise precision as the bit-wise optimization by reformulating the weight using the sum of the PoT basis (same as our binary representation of (4). Then they directly optimize each bit using a straight-through estimator (STE).

Ours and BSQ differ in their motivation and technique for inducing bit-level sparsity. The goal of BSQ is to learn the layer-wise precision targeting the hardware supporting the dense mixed-precision operation (e.g., GPUs); on the other hand, our goal is to realize an efficient $\texttt{mult}$-free network based on the proposed $\texttt{add-shift-add}$ with sparse $\texttt{add}$ targeting the hardware supporting the unstructured sparsity (e.g., data-flow processors, CPUs, FPGAs, etc.). Due to the difference in their goal, BSQ applies the sparsification for *groups of weights* (e.g., each layer), while we apply the sparsification for each *individual weight element*. In BSQ, sparsity is induced by *optimizing each bit independently* using STE, which may suffer from severe quantization error. In contrast, ours *optimize in the quasi-continuous weight space* using the proximal weight by utilizing the equivalent of the $\texttt{add-shift-add}$ and $\texttt{mult-add}$.