Automated Log-Scale Quantization for Low-Cost Deep Neural Networks

Sangyun Oh[1], Hyeonuk Sim[2], Sugil Lee[1], Jongeun Lee[1†]
[1]Department of Electrical Engineering, UNIST, Ulsan, Korea
[2]Department of Computer Science and Engineering, UNIST, Ulsan, Korea
{syoh,detective,sglee17,jlee}@unist.ac.kr

Abstract

Quantization plays an important role in deep neural network (DNN) hardware. In particular, logarithmic quantization has multiple advantages for DNN hardware implementations, and its weakness in terms of lower performance at high precision compared with linear quantization has been recently remedied by what we call selective two-word logarithmic quantization (STLQ). However, there is a lack of training methods designed for STLQ or even logarithmic quantization in general. In this paper we propose a novel STLQ-aware training method, which significantly outperforms the previous state-of-the-art training method for STLQ. Moreover, our training results demonstrate that with our new training method, STLQ applied to weight parameters of ResNet-18 can achieve the same level of performance as state-of-the-art quantization method, APoT, at 3-bit precision. We also apply our method to various DNNs in image enhancement and semantic segmentation, showing competitive results.

1. Introduction

Quantization plays an important role in implementing energy-efficient hardware for deep neural networks. Previous work on quantization mostly considers uniform quantization, but non-uniform quantization schemes such as logarithmic-scale quantization (log-scale quantization, for short) [19, 15] can have advantages over linear quantization in terms of hardware implementation, since it can replace multiplier hardware with a shifter or an adder, depending on whether log-scale quantization is applied to one or both operands.

Log-scale quantization places more quantization boundaries for low-magnitude values at the expense of less boundaries for high-magnitude values. This can sometimes lead to a lower expected quantization error overall, but often disproportionately high error that happens at high-magnitude values can undermine the accuracy of a log-scale quantization scheme significantly, resulting in worse performance than linear quantization at the same bit-width [15]. This problem of log-scale quantization has recently been addressed by selective two-word logarithmic quantization schemes (*e.g.* SLQ [16], FLightNN [9]). Selective two-word logarithmic quantization (STLQ) employs the vanilla log-scale quantization most of the time, but when the quantization error is higher than a certain threshold, the error is quantized again, emitting two log-quantized words. The de-quantization process is straightforward. Importantly, this scheme allows the same low-complexity hardware of log-scale quantization to be used while showing similar accuracy as linear quantization.

However, the previous work [16, 9] has important shortcomings. First, it uses hyperparameter(s) to control the ratio of two-word quantization, which is suboptimal if the same value is used for all layers/channels, or cumbersome to optimize if different values are used. Second, though some previous work [9] uses trainable latent variables to balance between accuracy and model sparsity through differentiable training, their optimization goal does not explicitly target model size or the number of two-word ratio, but instead some measure of quantization error, which is only loosely related to the two-word ratio. Consequently the training result is not optimal in terms of accuracy vs. two-word quantization ratio. Third, finding the best weight parameters for a given two-word quantization ratio, which is a very useful use case scenario, is not directly supported by previous work.

In this paper we propose a novel differentiable training framework for STLQ. Our method can directly optimize weights for a given two-word quantization ratio. Starting with a two-word ratio constraint has a surprising side effect that we use to our advantage in our training. That is, we can pre-select and separate the weight parameters that are likely to require two-word quantization from the rest of the

†Corresponding author.
This work was supported by Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd., IITP grant funded by MSIT of Korea (No.2020-0-01336, Artificial Intelligence Graduate School Program (UNIST)), and Free Innovative Research Fund of UNIST (1.170067.01).

weight parameters, and treat the two groups differently with different loss functions. We also propose per-tile quantization, which is to determine the number of quantized words per what is known as *weight tile*, based on the observation that the only requirement for efficient hardware implementation of STLQ is that all weight parameters within a weight tile be quantized with the same number of words.

These optimizations help us achieve significantly better performance than the state-of-the-art training method for STLQ [9], which is also based on differentiable training, and achieve close to floating-point baseline performance for ResNet models using 3-bit STLQ weight and 3-bit linear activation, with a low two-word ratio (5∼15%).

We also apply STLQ and our training method to more challenging networks and tasks such as MobileNetV2, ShuffleNetV2, image enhancement [2], and semantic segmentation [4], which typically require higher precision activation and weight. Our training result shows very competitive results at 3-bit. It is important to note that in all the experiments we use 3-bit STLQ quantization but sometimes higher two-word ratios to achieve good performance (for MobileNetV2 and ShuffleNetV2), which attests to the flexibility of the STLQ scheme.

We also compare against state-of-the-art quantization, APoT [17], which is more a novel quantizer than a training method. Our 3-bit ResNet results compare favorably against APoT in terms of accuracy vs. FixOPS (a measure of hardware complexity). Considering the advantages of STLQ such as the reusability of logarithmic hardware and flexibility to increase performance by adjusting two-word ratio, we believe that our training method for STLQ is a very valuable contribution.

We make the following contributions. First, we propose a novel training framework for STLQ. It takes a two-word quantization ratio as a constraint, and finds the best trained weights. Second, our training method significantly outperforms the previous state-of-the-art STLQ training method. Third, we have applied this method to various applications such as image classification, image enhancement, and semantic segmentation, which has shown competitive results.

2. Related Work

Previous studies introduced various quantization methods that utilize non-uniform distribution of weights. LogNet [15] proposed log-scale quantization and hardware for power-of-two values using shifter operation. Distillation [21] proposed training method that improves quantization performance by adding the quantization levels to the task loss and letting levels to be learned during training.

LQ-Nets [26] proposed method to learning the quantization levels from the quantization error by introducing Quantization Error Minimization (QEM). SLQ [16] proposed log-scale quantization using two words (one is optional), searched for a threshold value that minimizes the overhead for extra word, and proposed energy-efficient hardware as well. APoT [17] proposed a new quantizer that reduces the quantization error through an addition based representations of power-of-two values from predetermined bit sets and applies scalable parameters for weight and activation in training process.

Another studies introduced quantization-aware training (QAT). QAT have attempted to explore quantization-related learnable parameters in the training process. Usually these parameters correspond to small decimal values and jointly trained with weights. PACT [6] introduced decimal value called alpha to scale the clipping of activations which involved in training process. QIL [13] proposed training method to find the optimal quantization boundaries by learning the precision delta of linear-scale. FLightNN [9] proposed a method of finding the threshold value for quantization error through training and suggested a filter unit based hardware optimization.

3. Preliminaries on Log-Scale Quantization

3.1. Logarithmic Quantization

Consider an N-bit integer q, which is quantized to log-scale from a real number x in $[-1,1]$. The range of q is $[-M, M-1]$, where $M = 2^{N-1}$. Then the reconstructed value $\tilde{x}$ can be given as follows.

$$\text{LogDequant} : \tilde{x} = \begin{cases} 0 & \text{if } q = 0 \\ \text{sign}(q)\, 2^{-|q|} & \text{otherwise.} \end{cases} \tag{1}$$

We define log-scale quantizer as follows.

$$\text{LogQuant} : q = \begin{cases} \text{clip}\left(-\left\lfloor \log_2 \frac{|x|}{U} \right\rceil, 1, M-1\right) & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -\text{clip}\left(-\left\lfloor \log_2 \frac{|x|}{U} \right\rceil, 1, M\right) & \text{otherwise.} \end{cases}$$

where clip is the clipping function defined as $\text{clip}(x,a,b) = \min(\max(x,a),b)$, $\lfloor \rceil$ the round operation, and U a scaling factor (e.g., may be set to the maximum of $|x|$). Let LogQ denote the function from x to $\tilde{x}$, where $\circ$ is a function composition, simulating the effect of log-scale quantization.

$$\text{LogQ} : x \to \tilde{x} = \text{LogDequant} \circ \text{LogQuant} \tag{2}$$

3.2. Selective Two-word Logarithmic Quantization

Selective two-word logarithmic quantization (STLQ) applies log-scale quantization once or twice depending on the quantization error.

Round 1:

$$\tilde{x}_1 = \text{LogQ}(x), \qquad r_1 = x - \tilde{x}_1 \tag{3}$$

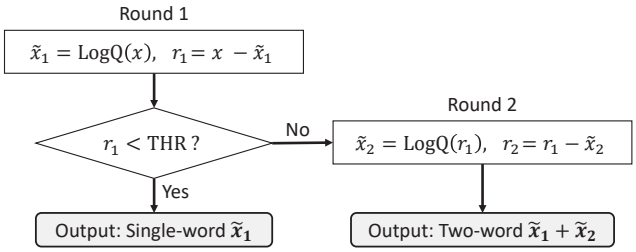

Figure 1. Selective two-word logarithmic quantization (STLQ).

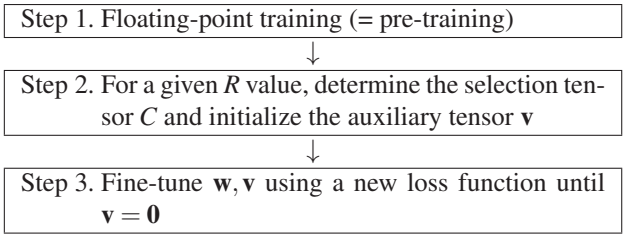

Figure 2. Overview of our training flow for STLQ.

Round 2:

$$\tilde{x}_2 = \text{LogQ}(r_1), \qquad r_2 = r_1 - \tilde{x}_2 \tag{4}$$

The quantizer emits either $\{\text{LogQuant}(x), \text{LogQuant}(r_1)\}$ if two-word quantization is used, or $\text{LogQuant}(x)$ otherwise.

To efficiently handle the variable number of quantized words, it has been suggested [9] that the granularity of choosing the number of quantized words, or *quantization cardinality granularity*, be a filter (of a weight tensor) rather than an individual element. In this case, filters whose average quantization error (*e.g.* $||\mathbf{r_1}||_2$) exceeds a certain threshold can be quantized with two words (per element).

4. Our Proposed Training Method

4.1. Budget-constrained Quantization

Unlike previous work, our training framework takes the ratio of two-word quantization R as input, and generates the best trained weight satisfying the constraint. In this section we assume that quantization cardinality granularity is a weight element. Later we extend it to a larger unit.

Our training method is based on two ideas. The first is *predestination*, meaning that since we know the ratio of two-word quantization R, we may better split the weight parameters into two sets, and apply different optimizations to each set. The selected weight parameters are trained to minimize task loss without any pressure on the residual while the unselected ones are trained to minimize the residual. The second is *auxiliary tensor* $\mathbf{v}$, which has the same dimension as the weight tensor, and models, for the unselected ones, the degree of transition from two-word quantization to single-word quantization. By making the auxiliary tensor an independent variable that is jointly trained with weight tensor, weight training can be decoupled from residual reduction.

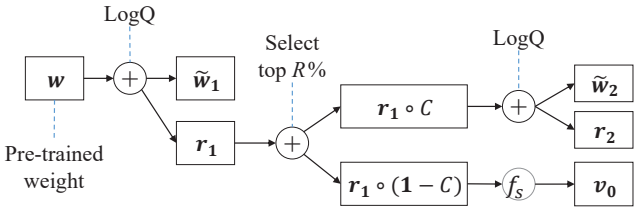

Figure 3. Determining the selection tensor C and initial value $\mathbf{v_0}$ of auxiliary tensor.

Figure 2 illustrates the overall flow of our training framework. First we train a network without quantization; the resulting weight is called pre-trained weight ($\mathbf{w}$). Second, from the pre-trained weight and two-word ratio R, we determine the selection tensor C, which partitions the weight parameters into two sets, as well as initialize the auxiliary tensor. Third we jointly train $\mathbf{w}$ and $\mathbf{v}$ until all unselected ones have transitioned to single-word quantization.

4.2. Selection Tensor

Figure 3 illustrates our selection and auxiliary tensor initialization algorithm. First we apply log-scale quantization to the pre-trained weight, and examine the residual $\mathbf{r_1}$ to decide which ones to select for two-word quantization.

$$\tilde{\mathbf{w}}_\mathbf{1} = \text{LogQ}(\mathbf{w}), \qquad \mathbf{r_1} = \mathbf{w} - \tilde{\mathbf{w}}_\mathbf{1} \tag{5}$$

We simply select the top R% elements based on the first round residual. The rationale is this: Since $\mathbf{w}$ is a pre-trained weight, and our fine-tuning will not disrupt $\mathbf{w}$ in a major way, it is reasonable to assume that elements with the greatest residual ($|r_1|$) are the ones that need second round quantization the most. Once selected, they always have two rounds of quantization; in other words, C is never updated.

$$C = \text{Select}(|\mathbf{r_1}|, R) \tag{6}$$

Selection tensor C is a binary tensor, not trainable, of the same size as $\mathbf{w}$. Its elements are 1 for selected weight parameters and 0 for the others. This selection is made on a per-layer basis (i.e., R% is chosen from each layer).

4.3. Fine-tuning and Initializing Auxiliary Tensor

Now the crucial question is how to remove second round quantization from the unselected elements without affecting the performance of a neural network. Rather than removing second round quantization all at once, which is very likely to leave an unrecoverable performance gap, we gradually phase out the effect of $\mathbf{r_1}$ by multiplying a scale factor to it. Moreover, we design the scale factor to be an independent variable $\mathbf{v}$ and train it via gradient descent along with $\mathbf{w}$ such that the task loss can be minimized. To encourage $\mathbf{v}$ to

decrease over time, we add a regularization term to the loss function.

Specifically, the new loss function is as follows:

$$\ell(\mathbf{w},\mathbf{v}) = \ell_{task}(\tilde{\mathbf{w}}) + \lambda||\mathbf{v}||_2^2 \quad (7)$$

$$\tilde{\mathbf{w}} = \tilde{\mathbf{w}}_\mathbf{1} + \mathrm{LogQ}(\mathbf{r}_\mathbf{1} \circ (C + \mathbf{v})) \quad (8)$$

where $\circ$ is Hadamard product, and λ is a weighting factor (hyperparameter) balancing the task loss and the new loss term. The auxiliary tensor $\mathbf{v}$ is complementary to C, meaning it has nonzero elements only where C is zero. For unselected elements, $\mathbf{v}$ models the fractional degree of transition from two-word quantization ($v = 1$) to single-word quantization ($v = 0$).

We initialize $\mathbf{v}$ to $\mathbf{v_0}$ below (though we believe other initialization schemes would also work), where f_s is a scale function that applies softmax (excluding the $C = 1$ elements) and normalization (i.e., divide by the max element) in sequence, so that the largest residual has $v = 1$. The effect of this initialization is that smaller (in magnitude) residuals will phase out first. Applying softmax to small values (note: $r_1 \ll 1$) reduces their differences, with the subsequent normalization pulling all values close to 1.

$$\mathbf{v_0} = f_s(|\mathbf{r}_\mathbf{1} \circ (\mathbb{1} - C)|) \quad (9)$$

$$f_s(\mathbf{x}) = \mathrm{normalize}(\mathrm{softmax}(\mathbf{x})) \quad (10)$$

The new term in the loss function plays the same role as weight decay for $\mathbf{v}$, but the gradient of $\mathbf{v}$, $\frac{\partial \ell}{\partial v}$, is also contributed by the task loss, since $\mathbf{v}$ is involved in the forward calculation (see (8)). As a result, the effect of second round quantization is gradually reduced in a manner that best minimizes the task loss. During training, we truncate the elements of $\mathbf{v}$ that are less than ε in magnitude, to prevent small fluctuations of $\mathbf{v}$ near zero (we set ε to 0.001 in our experiments), and also ensure that $\mathbf{v}$ remains complementary to C. After fine-tuning, the quantized weights are given as follows: $\{\tilde{w}_1, \tilde{w}_2\}$ for the chosen elements ($C = 1$), and $\tilde{w}_1$ for the others ($C = 0$).

4.4. Per-Tile Quantization

To address the hardware design issue caused by the varying number of quantized words among weight parameters, the previous work [9] suggests a per-filter scheme, i.e., decide the number of quantized words for each filter. While that can avoid the problem, it is an over-design for hardware accelerators. The root cause of the problem is that hardware accelerators often consist of a large array of processing elements (PEs), which can become out of synchronization if some PEs take two cycles while others take one, leading to significantly increased application latency. In short, the variable number of quantized words implies variable-cycle MAC (multiply-accumulate) operation, which is very inefficient for array processing.

Therefore to avoid the variable-cycle MAC problem, we only need to ensure that weight parameters used simultaneously have the same number of quantized words. Fortunately, the set of weight parameters used simultaneously, which is called *weight tile* or *tile*, is known at design time. For instance, a weight tile in DianNao [5] has the dimension $(T_n, T_m) = (16, 16)$, where T_n is the size along the input channels and T_m is the size along the output channels. Since a tile is typically much smaller than a filter, we can expect higher performance from a per-tile scheme.

Extending our training framework for per-tile quantization is straightforward. We assume that the tile dimension is known at training time. The main change in our framework is the size of tensor C, the elements of which now correspond to tiles in a weight tensor. Figure 5 shows an example of the DianNao's tile (T_n, T_m) applied to a convolution layer, where C and $\mathbf{v_0}$ now have the following dimension: $(\lceil \frac{M}{T_m} \rceil, \lceil \frac{N}{T_n} \rceil, K, K)$, where M and N are the number of output and input channels, respectively, and K is the size of a convolution filter. To rank tiles, we use the ℓ^2-norm of the first residual, $||\mathbf{r}_\mathbf{1}||_2$, where $\mathbf{r}_\mathbf{1}$ is of a weight tile, and not of the entire weight tensor.

4.5. Discussion: Differences from Previous Work

One minute difference from the previous work [9] is that while the previous work tries to minimize the number of quantized words for all elements, we do so for unselected ones only. In other words, we need not suppress second round quantization for the $C = 1$ elements (i.e., chosen elements), which helps us achieve high accuracy even with low R values.

Another difference is that instead of minimizing $\mathbf{r}_\mathbf{1}$ directly (*e.g.* by adding a penalty term for $||\mathbf{r}_\mathbf{1}||_2$ to the loss function), we introduce an independent variable $\mathbf{v}$ and regard the product $\mathbf{v} \circ \mathbf{r}_\mathbf{1}$ as the actual residual to be passed to the second round quantization. By doing so, we can ensure that the actual residual will converge to zero, irrespective of how $\mathbf{r}_\mathbf{1}$ or $\mathbf{w}$ changes during training, which can lead to faster convergence.

Lastly, we determine the number of quantized words on a per-tile basis as opposed to on a per-filter basis.

5. Experimental Setup

To evaluate the effectiveness of our training method, we use various datasets and convolutional neural network (CNN) models in three applications: image classification, image enhancement, and semantic segmentation. In all our

Also slightly larger model size due to increased meta data describing the number of quantized words per tile.

Note that we use the same quantization parameters such as scale factor U within a layer. In that sense, our per-tile quantization is still per-layer quantization; only the number of quantized words can be different at the granularity of a tile.

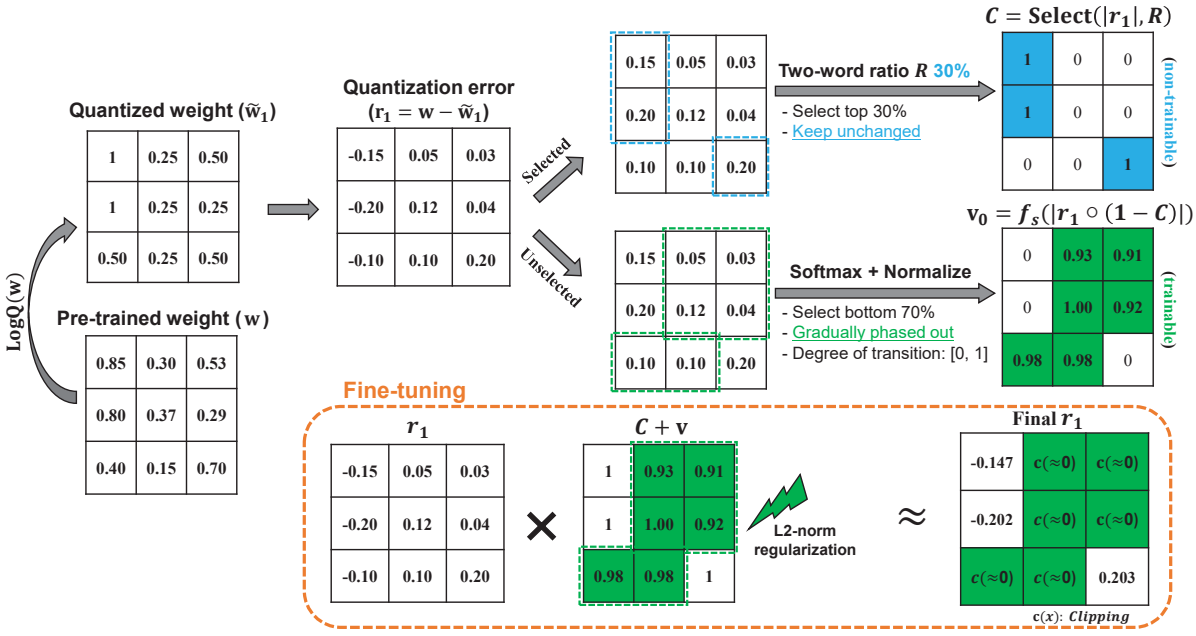

Figure 4. Example for our training flow ($R = 30\%$).

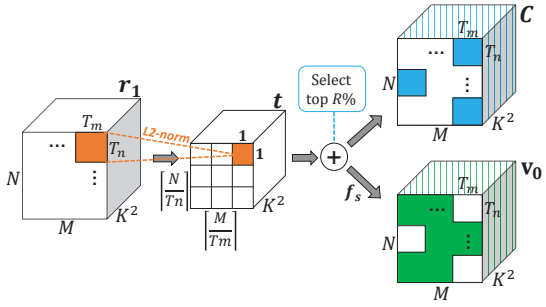

Figure 5. Per-tile quantization example for a convolution layer.

Table 1. Networks from [9], which are the largest models using relatively large datasets. **Layers** is the total number of convolution layers and **Filters** is the maximum channel depth of the network.

Network	Params	Base Model	Layers	Filters	Dataset
ResNet_A	0.7M	ResNet-18	18	128	CIFAR-100
ResNet_B	2.8M	ResNet-18	18	256	CIFAR-100
ResNet_C	1.8M	ResNet-18	10	256	ImageNet

experiments we quantize all convolution (including deconvolution) layers. All our training was performed using PyTorch 1.6.0 [1] on Python 3.6 running on NVIDIA RTX 2080Ti GPU with CUDA 9.0 and cuDNN 7.3.0.

5.1. Image Classification

We use 8 CNNs from image classification, including three modified ResNet-18 models (ResNet A, B, C), three original ResNet models (ResNet-18, ResNet-34, ResNet-50) [11] and two well-known light-weight models, MobileNetV2 [23] and ShuffleNetV2 [18].

ResNet A, B, and C are included to provide a direct comparison with the FLightNN results [9]. They are the three largest models in their results, and we use the same setting as in the previous work. Details of the models are summarized in Table 1. The other five models are used without modification, and trained with ImageNet dataset. MobileNetV2 and ShuffleNetV2 have scaling parameters for controlling model complexity, which are set to the default value, 1.0.

5.1.1 Training of Unmodified Models

For the five unmodified models, we used a pre-trained weight from the Torchvision models provided by PyTorch [1] and used SGD (Stochastic Gradient Descent) optimizer [24] with momentum 0.9 and weight decay 1e-6, and LR (Learning Rate) is set to 1e-4 with cosine decay scheduler for fine-tuning cases which performing 10 epochs.

5.1.2 Training of Modified ResNet Models

For ResNet A and B, we trained the full precision (FP) baseline models from scratch. The same training protocol was for both A and B: a total of 200 epochs are performed using SGD optimizer with momentum 0.9 and weight decay 5e-4. LR started at 0.1, and gamma for LR decay is set to 0.2 and applied to 60, 120, and 160 epoch.

ResNet C was also trained from scratch in FP, for a total of 120 epochs. We used SGD optimizer with momentum 0.9 and weight decay 5e-4. LR started at 0.01 and we used a step scheduler with LR gamma 0.1, which was applied to 60, 90, and 120 epoch.

Table 2. Quantization performance results for CIFAR-100 dataset. For fair comparison, we manually trained baseline FP (full precision) models from scratch which have similar performance of reference FP cases. L_x indicates latent variables for x case in Method [9]. Our all cases have same Select Ratio 5% and Per-Tile Quantization (PTQ) is enabled. Tile size is 16×16.

Model	Method [9]	Bit Width (W-A)	Top-1 (%)	Storage (MB)
ResNet_A	FP	32-32	69.16	2.80
	LightNN-2	8-8	68.84	0.70
	LightNN-1	4-8	67.32	0.40
	Fixed Point	4-8	67.67	0.40
	FLightNN_a	L_a-8	68.59	0.40
	FLightNN_b	L_b-8	68.76	0.60
	Ours	32-32 (FP)	69.15	2.80
		4-8	71.16	0.41
		3-6	70.93	0.28
		3-5	70.89	0.28
		3-4	70.56	0.28
		3-3	69.31	0.28
ResNet_B	FP	32-32	71.22	11.20
	LightNN-2	8-8	70.96	2.80
	LightNN-1	4-8	69.71	1.40
	Fixed Point	4-8	69.34	1.40
	FLightNN_a	L_a-8	70.85	1.40
	FLightNN_b	L_b-8	70.87	2.40
	Ours	32-32 (FP)	71.20	11.20
		4-8	73.43	1.43
		3-6	73.37	1.12
		3-5	73.21	1.12
		3-4	73.14	1.12
		3-3	71.62	1.12

Table 3. Quantization performance results for ImageNet dataset. L_x indicates latent variables for x case in Method [9]. Our all cases have the same Select Ratio 5% and Per-Tile Quantization (PTQ) is enabled. Tile size is 16×16.

Model	Method [9]	Bit Width (W-A)	Top-5 (%)	Storage (MB)
ResNet_C	LightNN-2	8-8	75.04	1.80
	LightNN-1	4-8	72.94	0.90
	FLightNN_a	L_a-8	74.80	1.50
	FLightNN_b	L_b-8	75.00	1.70
	Ours	32-32 (FP)	79.52	3.79
		4-8	79.76	1.52
		3-6	79.71	1.27
		3-5	79.39	1.27
		3-4	78.02	1.27
		3-3	74.93	1.27

5.2. Image Enhancement

Image enhancement and semantic segmentation require higher-precision activation compared with image classification, thus can be a better fit with logarithmic quantization. For image enhancement, we use the SID network [2], which adopts U-Net [22] as the default backbone architecture. The U-Net model is an encoder-decoder network consisting of 19 convolutional layers and 4 deconvolutional layers. We use the Sony dataset, which is a set of images taken at 4240×2832 resolution through the Sony $\alpha 7S$-II camera that has a full-frame Bayer sensor. It has three categories according to the exposure time, which contain 1190, 699, and 808 images each, for a total of 2697 images. 10% of these images selected randomly are used as the validation set.

For training, we used the pre-trained weight and training protocol published by the authors. We used Adam optimizer [14] with momentum 0.9 and input is randomly cropped to 512×512 in each training iteration. We set the LR to 1e-5 for fine-tuning.

5.3. Semantic Segmentation

We used the network (and the pre-trained weight) from DeepLabV3+ [4], which has various backbone CNN encoder options. We chose ResNet-101, MobileNetV2 and DRN [25] as our baseline encoders. We fine-tuned those model variations with the PASCAL VOC 2012 [10] dataset, which provides annotations for 20 foreground objects and one background class as well as 10582 images for training and 1449 images for validation.

We followed the training protocol from [3] and the poly learning scheduler from [20] with the LR 7e-4, cropped to 513×513 with batch normalization layers [12], which ran for 10 epochs for fine-tuning.

6. Experimental Results

6.1. Comparison with FLightNN

To compare with FLightNN [9], the state-of-the-art training method for STLQ, we use the three modified ResNet models. First, since there is no official pre-trained weight, we prepared baseline models through scratch training (see Section 5.1). For fair comparison, we confirmed that the baselines have almost the same performance as the FLightNN's. All our results are fine-tuned by adding 10 epochs to the end point of the baseline training and gamma for LR is 1.0. In addition, we applied 5% select ratio and enabled Per-Tile Quantization (PTQ) in all cases.

In Table 2 and Table 3, we compared the results at the same bit precision as of FLightNN, as well as lower precision settings. In the tables, *FP* means floating-point, *Fixed Point* linear quantization, and *Storage* the capacity to be stored as an actual model file, which is the metric used in the previous work (LightNN models are a previous version of FLightNN).

The table clearly indicates that our method performs significantly better than FLightNN at the same precision set-

https://github.com/cchen156/Learning-to-See-in-the-Dark

Table 4. Select ratio ablation study for various applications on the validation test. We measure Top-1 accuracy of FP (full precision) cases by PyTorch official models [1]. Quantization is applied to all convolutional layers. $STLQ_{max}$ (Select Ratio 100%) is the STLQ case with all weight parameters quantized to two words. We used ImageNet [8] dataset for Image Classification, Sony dataset [2] for Image Enhancement, and PASCAL VOC 2012 [10] for Semantic Segmentation.

Image Classification	Method	Bit Width (W-A)	Top-1 (%)	Select Ratio (%)	Perf. Diff (%)	Params (M)
ResNet-18 [11]	Baseline	32-32 (FP)	69.75	N/A		11.68
	$STLQ_{max}$	5-5	69.74	100	(0.01)	4.31
	Proposed	5-5	69.72	15	(0.03)	1.73
		3-3	69.30	15	(0.45)	1.73
			69.28	10	(0.47)	1.66
			69.19	5	(0.56)	1.59
ResNet-34 [11]	Baseline	32-32 (FP)	73.30	N/A		21.78
	$STLQ_{max}$	5-5	73.13	100	(0.17)	8.10
	Proposed	5-5	73.08	15	(0.22)	3.19
		3-3	73.01	15	(0.29)	3.19
			72.87	10	(0.43)	3.05
			72.65	5	(0.65)	2.92
ResNet-50 [11]	Baseline	32-32 (FP)	76.13	N/A		25.50
	$STLQ_{max}$	5-5	75.70	100	(0.43)	9.31
	Proposed	5-5	75.56	15	(0.57)	3.88
		3-3	74.38	15	(1.75)	3.88
			74.30	10	(1.83)	3.74
			74.15	5	(1.98)	3.59
MobileNetV2 [23] (depth scale: $\times 1.0$)	Baseline	32-32 (FP)	71.87	N/A		3.47
	$STLQ_{max}$	3-6	69.10	100	(2.77)	0.87
	Proposed	3-6	68.92	50	(2.95)	0.73
			68.80	40	(3.07)	0.70
			68.55	35	(3.32)	0.69
ShuffleNetV2 [18] (width scale: $\times 1.0$)	Baseline	32-32 (FP)	69.36	N/A		2.26
	$STLQ_{max}$	3-6	66.46	100	(2.90)	0.57
	Proposed	3-6	66.38	30	(2.98)	0.47
			66.14	25	(3.22)	0.46
			66.05	20	(3.31)	0.45

Image Enhancement	Method	Bit Width (W-A)	PSNR (dB)	Select Ratio (%)	Perf. Diff (%)	Params (M)
SID [2] (backbone: U-Net [22])	Baseline	32-32 (FP)	28.59	N/A		7.76
	$STLQ_{max}$	3-6	27.80	100	(0.79)	1.94
	Proposed	3-6	27.78	15	(0.81)	1.12
			27.74	10	(0.85)	1.07
			27.67	5	(0.92)	1.02

Semantic Segmentation	Method	Bit Width (W-A)	mIoU (%)	Select Ratio (%)	Perf. Diff (%)	Params (M)
DeepLabV3+ [4] (backbone: ResNet-101 [11])	Baseline	32-32 (FP)	78.37	N/A		59.23
	$STLQ_{max}$	3-6	78.22	100	(0.15)	14.81
	Proposed	3-6	78.16	15	(0.21)	10.31
			78.08	10	(0.29)	10.04
			77.96	5	(0.41)	9.78
DeepLabV3+ [4] (backbone: MobileNetV2 [23])	Baseline	32-32 (FP)	71.24	N/A		5.78
	$STLQ_{max}$	3-7	69.14	100	(2.10)	1.45
	Proposed	3-7	68.76	50	(2.48)	1.33
			68.72	45	(2.52)	1.32
			68.58	40	(2.66)	1.31
DeepLabV3+ [4] (backbone: DRN [25])	Baseline	32-32 (FP)	79.16	N/A		40.68
	$STLQ_{max}$	3-6	79.03	100	(0.13)	10.17
	Proposed	3-6	78.87	15	(0.29)	7.97
			78.72	10	(0.44)	7.84
			78.70	5	(0.46)	7.71

ting, as well as having the same level of storage (model size). In many cases, our lower precision results are often better than the result of the previous method, with a significant margin. This result clearly shows the superiority of our training method.

6.2. Ablation Study for Various Select Ratios

We used the official pre-trained weights for ablation study, and ran fine-tuning for 10 epochs. As previously mentioned, we did not change the training hyperparameters, but only the scheduler option so that additional epochs can be performed. We compare mainly against STLQ_{max}, which is the upper bound to our STLQ training performance.

The purpose of our training method is to optimize the model to predetermined select ratio while maintaining a similar level of performance as STLQ_{max} as much as possible. Therefore, we first obtained STLQ_{max} that has minimum combination of weight-activation bits and a small performance degradation from baseline. In this case, we were able to obtain models with a performance degradation of around 1% at weight 3-bit and activation 3-6 bit in most applications.

However, in the case of image enhancement and semantic segmentation which use relatively high-resolution images, the minimum required activation bit was higher (6-7 bits) while 3-bit weight was sufficient. Another phenomenon is that in image classification, light-weight models [23, 18] require more activation bits as well. In this case, 6-bit activation was required even at the performance degradation of up to 3%, so we selected 3-bit weight and 6-bit activation for STLQ_{max}.

The training results are summarized in Table 4, which shows that our training method can often achieve performance close to the floating-point baseline, and generally has very small differences from the STLQ_{max} results even at 5% two-word ratio. One exception is the light-weight models, which require higher two-word ratios of up to 50%. This is caused by the depthwise separable convolution [7] layers which already reduce the model size significantly. On the other hand, our result on light-weight models demonstrates that the STLQ scheme powered by our training method can be used flexibly to handle various kinds of workloads, by just varying the two-word ratio without changing the base precision (3-bit).

6.3. Comparison with APoT Quantization

We also compare our training results with that of APoT [17], which is more a novel quantizer rather than a training method, though it also includes training techniques. There are a number of differences between the two approaches. APoT uses a customized quantizer that is a hybrid between linear and STLQ quantizers. Also APoT is applied to both weight and activation whereas we apply STLQ to weight only (activation is linear quantized).

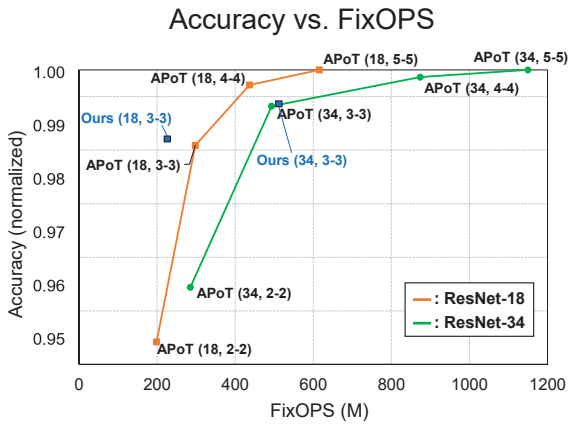

Figure 6. Accuracy-FixOPS comparison with APoT quantization.

Figure 6 shows the result. Since the baseline (floating-point) accuracy of APoT is different from that of ours, we use normalized accuracy, which is the the validation accuracy divided by the floating-point accuracy. The FixOPS is the metric suggested by APoT [17]. The two-word ratio of our models (for both) is 15%.

The graph shows that our 3-bit training results compare favorably against APoT. In fact, our method outperforms 3-bit APoT in both accuracy and FixOPS in the case of ResNet-18. In the case of ResNet-34, ours has slightly better accuracy with slightly higher FixOPS.

Considering certain advantages stemming from the fact that STLQ uses essentially the same hardware as logarithmic quantization hardware but is equipped with the flexibility to easily boost accuracy by modulating two-word ratio (as partially demonstrated by our MobileNetV2 and ShuffleNetV2 result), we believe that our training method for STLQ is a very valuable contribution.

7. Conclusion

We proposed a new training method that minimizes the overhead of current STLQ-based state-of-the-art techniques [16, 9] and performs significantly better at low precision via a fully automated approach. This is enabled by exploiting the two-word ratio constraint as well as a finer-granularity of deciding the number of quantized words. Through our experiments we showed that our method can achieve the same level of performance as state-of-the-art quantization method (APoT [17]) at 3-bit precision, in addition to performing significantly better than the previous differentiable training-based method for STLQ. An important advantage of STLQ is that its performance can be modulated not only by the base precision but also by the two-word quantization ratio, which can be exploited *e.g.* when designing a single hardware architecture running various models with different levels of challenge/complexity, which is left for future work.

References

[1] *Pytorch: An open source machine learning framework*, 2020.

[2] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv, arXiv:1706.05587*, 2017.

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *Eur. Conf. Comput. Vis. (ECCV)*, 2018.

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014.

[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, arXiv:1805.06085*, 2018.

[7] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2017.

[8] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei. *ImageNet: Large Scale Visual Recognition Challenge (ILSVRC)*, 2012.

[9] Ruizhou Ding, Zeye Liu, Ting-Wu Chin, Diana Marculescu, and R. D. (Shawn) Blanton. Flightnns: Lightweight quantized deep neural networks for fast and accurate inference. *Proc. of the 56th Annual ACM/IEEE Design Automation Conference (DAC)*, 2019.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge (VOC)*, 2012.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2016.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015.

[13] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Youngjun Kwak, Jae-Joon Han, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Int. Conf. Learn. Represent. (ICLR)*, 2015.

[15] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. Lognet: Energy-efficient neural networks using logarithmic computation. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[16] Sugil Lee, Hyeonuk Sim, Jooyeon Choi, and Jongeun Lee. Successive log quantization for cost-efficient neural networks using stochastic computing. *Proc. of the 56th Annual ACM/IEEE Design Automation Conference (DAC)*, 2019.

[17] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *Int. Conf. Learn. Represent. (ICLR)*, 2020.

[18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *Eur. Conf. Comput. Vis. (ECCV)*, 2018.

[19] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *CoRR, abs/1603.01025*, 2016.

[20] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *Int. Conf. Comput. Vis. (ICCV)*, 2015.

[21] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv, arXiv:1802.05668*, 2018.

[22] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent (MICCAI)*, 2015.

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018.

[24] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013.

[25] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2017.

[26] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. *Eur. Conf. Comput. Vis. (ECCV)*, 2018.