

SQuantizer: Simultaneous Learning for Both Sparse and Low-precision Neural Networks

Mi Sun Park Xiaofan Xu Cormac Brick
Movidius, AIPG, Intel

mi.sun.park@intel.com, xu.xiaofan@intel.com, cormac.brick@intel.com

Abstract

Deep neural networks have achieved state-of-the-art accuracies in a wide range of computer vision, speech recognition, and machine translation tasks. However the limits of memory bandwidth and computational power constrain the range of devices capable of deploying these modern networks. To address this problem, we propose SQuantizer, a new training method that jointly optimizes for both sparse and low-precision neural networks while maintaining high accuracy and providing a high compression rate. This approach brings sparsification and low-bit quantization into a single training pass, employing these techniques in an order demonstrated to be optimal.

Our method achieves state-of-the-art accuracies using 4-bit and 2-bit precision for ResNet18, MobileNet-v2 and ResNet50, even with high degree of sparsity. The compression rates of $18\times$ for ResNet18 and $17\times$ for ResNet50, and $9\times$ for MobileNet-v2 are obtained when SQuantizing¹ both weights and activations within 1% and 2% loss in accuracy for ResNets and MobileNet-v2 respectively. An extension of these techniques to object detection also demonstrates high accuracy on YOLO-v2. Additionally, our method allows for fast single pass training, which is important for rapid prototyping and neural architecture search techniques.

Finally extensive results from this simultaneous training approach allows us to draw some useful insights into the relative merits of sparsity and quantization.

1. Introduction

High-performing deep neural networks [11, 14, 27] consist of tens or hundreds of layers and have millions of parameters requiring billions of float point operations (FLOPs). Despite the popularity and superior performance, those high demands of memory and computational power make it difficult to deploy on resource-constrained edge de-

vices for real-time AI applications like intelligent cameras, drones, autonomous driving, augmented and virtual reality (AR/VR) in retail, and smart healthcare. To overcome this limitation, academia and industry have investigated network compression and acceleration in various directions towards reducing complexity of networks, and made tremendous progresses in this area. It includes network pruning [9, 7, 30], network quantization [20, 39, 3], low-rank approximation [29, 33], efficient architecture design [27, 34], neural architecture search [40, 6] and hardware accelerators [8, 23].

In this work, we focus on two popular network compression techniques: network pruning and network quantization. More specifically, our focus is to unify the two techniques into a single training pass that jointly optimizes for both sparse and low-precision while achieving high accuracy and high compression rate. A few previous works [9, 19] applied both the techniques one after the other to show that pruning and 8-bit weight quantization can work together to achieve higher compression. However, applying one after the other not only requires two-stage training, but also makes it difficult to quantize with lower precision after pruning, due to the lack of understanding the impact of pruning weights on quantization, and vice versa. We therefore aim for more efficient network training process with both sparse low-precision weights and sparse low-precision activations.

This paper makes the following three contributions: first, we propose a new training method to enable simultaneous learning for sparse and low-precision neural networks that sparsify and quantize both weights and activations with low precision at each iteration of training. This method significantly reduces model size and computational cost but maintain high accuracy, therefore allows fast inference on resource-constrained edge devices.

Second, we analyze the order effect of the two compression techniques of sparsification and low-precision quantization when trained together. For example, should we first sparsify and then quantize the sparsified weights during each forward pass, or it should be the other way around

¹SQuantizing: joint optimization of Sparsification and low-precision Quantization.

to be more effective? After the analysis, we introduce our sparsification and quantization functions that can work better when simultaneously applied.

Third, we extensively evaluate the effectiveness of our approach on ImageNet classification task using ResNet18, ResNet50 and MobileNet-v2, and also extend to object detection task using YOLO-v2. We compare our method with several state-of-the-art quantization methods and outperform these methods for both 4-bit and 2-bit precision networks, even with high degree of sparsity.

To the best of our knowledge, this paper is the first to present an approach to simultaneous learning for both sparse and low-precision (weights and activations) neural networks, and achieve both high accuracy and high compression rate.

2. Related Work

Network Pruning: With a goal of easy deployment on embedded systems with limited hardware resources, substantial efforts have been made to reduce network size by pruning redundant connections or channels from pre-trained models, and fine-tune to recover original accuracy. While many of the related approaches differ in the method of determining the importance of weights or channels, the basic goal of removing unimportant weights or channels from original dense models to generate sparse or compact models remains the same. Specifically, fine-grained weight pruning (aka sparsification) [31, 10, 9, 7, 28] seeks to remove individual connections, while coarse-grained pruning [13, 16, 18, 22, 32] seeks to prune entire rows/columns, channels or even filters.

Deep compression [9] introduced three-stage training of pruning, trained quantization and Huffman coding and showed that weight pruning and weight quantization can work together for higher compression. Dynamic network surgery [7] presented an idea of connection splicing to restore incorrectly pruned weights from the previous step. Energy-aware pruning [31] proposed layer-by-layer energy-aware weight pruning for energy-efficient CNNs that directly uses the energy consumption of a CNN to guide the pruning process.

ThiNet [16] presented a filter level pruning method for modern networks with consideration of special structures like residual blocks in ResNets. AMC [12] introduced AutoML for structured pruning using reinforcement learning to automatically find out the effective sparsity for each layer and remove input channels accordingly. Generally speaking, coarse-grained pruning results in more regular sparsity patterns, making it easier for deployment on existing hardware for speedup but more challenging to maintain original accuracy. On the other hand, fine-grained weight pruning results in irregular sparsity patterns, requiring sparse matrix support but relatively easier to achieve higher sparsity.

Network Quantization: Network quantization is another popular compression technique to reduce the number of bits required to represent each weight (or activation value) in a network. Post-training quantization and training with quantization are two common approaches in this area. Post-training quantization is to quantize weights to 8-bit or higher precision from a pre-trained full-precision network with and without fine-tuning. Google Tensorflow Lite ² and Nvidia TensorRT ³ support this functionality by importing a pre-trained full-precision model and converting to 8-bit quantized model.

Significant progress has recently been made on training with quantization approach for low-precision networks [37, 38, 35, 39, 2, 36]. DoReFa-Net [37] presented a method to train CNNs that have low-precision weights and activations using low-precision parameter gradients. TTQ [38] introduced a new training method for ternary weight networks with two learnable scaling coefficients for each layer. INQ [35] proposed an incremental training method of converting pre-trained full-precision network into low-precision version with weights be either powers of two or zero. More recent researches [20, 39, 3] have tackled the difficulty of training networks with both low-precision weights and low-precision activations. Apprentice [20] used knowledge distillation technique by adding an auxiliary network to guide for improving the performance of low-precision networks. Low-bit CNN [39] presented three practical methods of two-stage optimization, progressive quantization and guided training with full-precision network. PACT [3] proposed a new activation quantization function with a learnable parameter for clipping activations for each layer.

A few previous works [9, 19] have applied both weight sparsification and weight quantization one after the other to show that pruning and 8-bit weight quantization can work together to achieve higher compression. Different from them, our method sparsifies and quantizes both weights and activations with low-precision (4-bit and 2-bit) by simultaneously learning in a unified single training while achieving high accuracy and high compression rate.

3. Our Method

In this section, we first introduce our SQuantizer for simultaneous learning for sparse and low-precision neural networks, and analyze the order effect of sparsification and quantization techniques when trained together. Furthermore, we elaborate on the details of our sparsification and quantization methods.

²https://www.tensorflow.org/lite/performance/post_training_quantization

³<https://developer.nvidia.com/tensorrt>

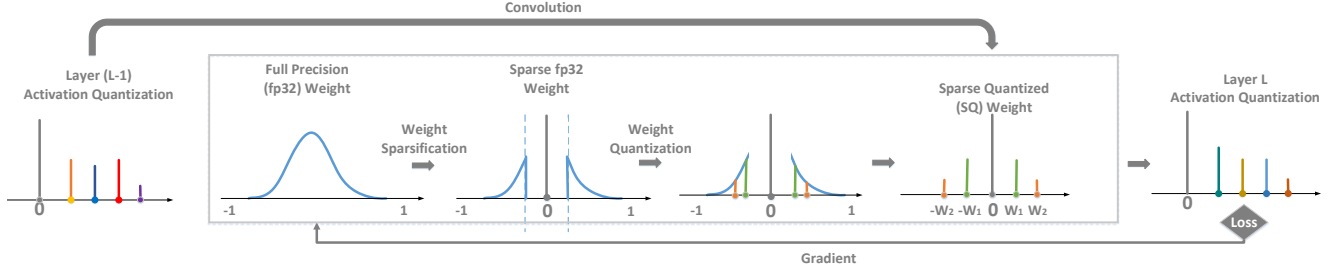


Figure 1. Overview of our proposed SQuantizer procedure for sparse and low-precision networks. The gray box shows the weight sparsification and quantization (SQuantization) steps in layer l in forward pass. The resulting sparse and low-bit quantized weights in layer l are then convolved with low-bit quantized output activations of the previous layer $l - 1$. In backward pass, the full-precision weights are updated with the gradients of SQuantized weights and activations at each iteration of training. Best view in color.

3.1. Learning both sparse and low-precision values

Our proposed SQuantizer method is illustrated in Figure 1. In each forward pass of training, we first sparsify full-precision weights based on a layer-wise threshold that is computed from the statistics of the full-precision weights in each layer. Then, we quantize the non-zero elements of the sparsified weights with min-max uniform quantization function (i.e. The minimum and maximum values of the non-zero weights). In case of activation, prior sparsification is not necessary, since output activations are already sparse due to the non-linearity of ReLU-like function. In general, ReLU activation function can result in about 50% sparsity. Therefore, we only quantize the output activations after batch normalization and non-linearity, which is also the input activations to the following convolutional layer.

In backward pass, we update the full-precision dense weights with the gradients of the sparse and low-bit quantized weights and activations. The gradients calculation for the non-differential quantization function is approximated with the straight-through estimator (STE) [1]. Our method allows dynamic sparsification assignment and quantization values by leveraging the statistics of the full-precision dense weights in each iteration of training.

After training, we discard the full-precision weights and only keep the sparse and low-bit quantized weights to deploy on resource-constrained edge devices. It is worth noting that, in case of activation quantization, we still need to perform on-the-fly quantization on output activations at inference time because it is dynamic depending on input images, unlike weights.

3.2. Analysis of the order effect

We analyze the order effect of the two compression (sparsification and quantization) techniques when applied together. Fundamentally, we want to find the optimal order that possibly leads to better performance. The two candidates are quantization followed by sparsification (S on Q), and sparsification followed by quantization (Q on S).

Figure 2 shows the weight histograms of layer3.5.conv2 layer (the last 3×3 convolutional layer in ResNet56) before and after applying the two techniques based on the two different orders. The top S on Q subfigure shows three histograms of full-precision baseline, 4-bit quantized weights, and sparse and 4-bit quantized weights (after sparsifying the quantized weights) respectively. The bottom Q on S subfigure represents the histograms of baseline, sparse weights, and sparse and 4-bit quantized weights (after quantizing the sparsified weights) respectively.

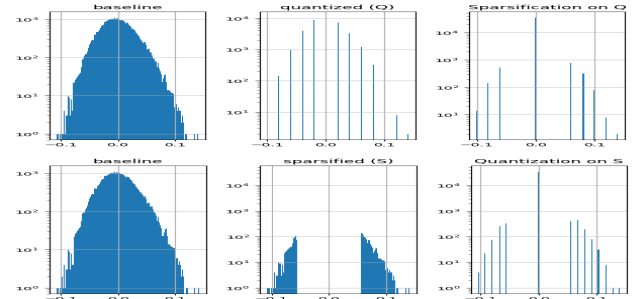


Figure 2. Weight histogram of ResNet56 layer3.5.conv2 layer on Cifar-10 using two different orders (S on Q and Q on S). X-axis are weight values and Y-axis are frequency at log-scale.

It is observed from the histograms of S on Q , we don't fully utilize all quantization levels. Although we can use up to 2^4 levels for 4-bit quantization, we end up using fewer levels due to the subsequent sparsification. The higher sparsity, the more number of quantization levels will be underutilized. Noting that this phenomenon heavily depends on sparsification methods. With random sparsification, you may still utilize all the levels and in this case, the order doesn't matter. However, magnitude-based methods [10, 7, 30] are commonly used and work better in practice. In fact, with Q on S , magnitude-based methods reduce the dynamic range of weights, thus reduce quantization error with finer quantization steps.

We also conduct experiments to verify our analysis with

ResNet56 on Cifar-10, as shown in Table 1. As expected, *Q on S* consistently outperforms *S on Q* in all three experiments. Moreover, our sparse and 4-bit quantized models give slightly better accuracy than the baseline, possibly because SQuantization acts as additional regularization which helps prevent overfitting. In summary, we believe that *Q on S* is more effective than *S on Q*, therefore we use *Q on S* in our SQuantizer.

Table 1. Validation accuracy of sparse and 4-bit quantized ResNet56 on Cifar-10 using two different orders (*S on Q* vs *Q on S*). W and A represent weight and activation respectively.

	<i>S on Q</i>		<i>Q on S</i>	
	Top1 (%)	Sparsity (%)	Top1 (%)	Sparsity (%)
baseline	93.37	-	93.37	-
sparse (4W, 32A)	93.42	57	93.45	57
sparse (4W, 32A)	93.34	73	93.40	73
sparse (4W, 4A)	92.88	57	92.94	57

3.3. SQuantizer in details

3.3.1 Sparsification

As shown in Figure 1, we first apply statistic-aware sparsification to prune connections in each layer by removing (zeroing out) the connections with absolute weight values lower than a layer-wise threshold. The basic idea of statistic-aware sparsification is to compute a layer-wise weight threshold based on the current statistical distribution of the full-precision dense weights in each layer, and mask out weights that are less than the threshold in each forward pass. In backward pass, we also mask out the gradients of the sparsified weights with the same mask.

We use layer-wise binary $mask_l^n$ (same size as weight W_l^n) for l^{th} layer at n^{th} iteration in Equation 1 and Equation 2. Similar to [7, 30], this binary mask is dynamically updated based on a layer-wise threshold and sparsity controlling factor σ (same for all layers). The mean and one standard deviation (std) of the full-precision dense weights in each layer are calculated to be a layer-wise threshold. This allows previously masked out weights back if it becomes more important (i.e. $|W_l^n(i, j)| > t_l^n$). It should be noted that our approach doesn't need layer-by-layer pruning [10, 31]. It globally prunes all layers with layer-wise thresholds considering different distribution of weights in each layer. Our experiment shows that our statistics-aware method performs better than globally pruning all layers with the same sparsity level, and is comparable to layer-by-layer pruning with much less training epochs.

$$mask_l^n(i, j) = \begin{cases} 0 & \text{if } |W_l^n(i, j)| < t_l^n \\ 1 & \text{if } |W_l^n(i, j)| > t_l^n \end{cases} \quad (1)$$

$$t_l^n = mean(|W_l^n|) + std(|W_l^n|) \times \sigma \quad (2)$$

Sparsity controlling factor σ is a hyper-parameter in this statistic-aware pruning method. Unlike explicit level of target sparsity (i.e. prune 50% of all layers), σ is implicitly determining sparsity level. To understand the effect of σ on accuracy and sparsity level, we experiment for 4-bit and 2-bit ResNet50 on ImageNet, shown in Table 2 and Figure 3. As expected, the higher σ , the more sparsity we can get with a slight decrease in accuracy. We can achieve up to $30\times$ compression rate for sparse and 4-bit model within 1% drop in accuracy, while achieving up to $42\times$ compression rate for sparse and 2-bit model within 2% drop in accuracy.

Table 2. Effect of various σ on accuracy and sparsity of ResNet50 on ImageNet. W represents weight.

	σ	Accuracy (%)		Sparsity (%)	#Params (M)	Compression Rate
		Top1	Top5			
baseline		76.3	93.0	0	25.5	-
4W	0.0	76.0	92.9	55	11.5	17 \times
	0.2	76.0	92.6	63	9.5	21 \times
	0.4	75.4	92.5	69	7.9	25 \times
	0.6	75.3	92.3	74	6.6	30\times
2W	0.0	74.8	92.2	56	11.3	36 \times
	0.1	74.6	92.1	59	10.4	39 \times
	0.2	74.5	92.0	63	9.5	42\times

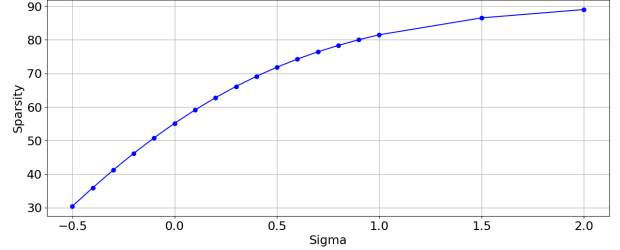


Figure 3. σ vs sparsity of sparse and 4-bit ResNet50. X-axis is sigma (σ) and Y-axis is sparsity level (%).

3.3.2 Quantization

After masking out relatively less important weights, we quantize the non-zero elements of sparsified weights with low-bitwidth k , as shown in Figure 1. For weight quantization, we use min-max uniform quantization function without clipping to $[-1, 1]$. Our min is the previously determined layer-wise pruning threshold t_l^n , while the max is the maximum value of the sparse weights $sparseW_l^n$ in l^{th} layer at n^{th} iteration of training. Based on Equation 3 to Equation 6, we quantize a full-precision non-zero element of sparse weight $sparseW_l^n(i, j)$ to k -bit w_{sq} .

$$max = max(|sparseW_l^n|), \quad min = t_l^n \quad (3)$$

$$w_s = \frac{|sparseW_l^n(i, j)| - min}{max - min} \quad (4)$$

$$w_q = \frac{1}{2^{k-1}-1} round((2^{k-1}-1)w_s) \quad (5)$$

$$w_{sq} = \text{sign}(\text{sparse}W_l^n(i, j))(w_q(\text{max} - \text{min}) + \text{min}) \quad (6)$$

In backward pass, in order to back-propagate the non-differentiable quantization functions, we adopt the straight-through estimator (STE) approach [1]. To be more specific, we approximate the partial gradient $\frac{\partial w_q}{\partial w_s}$ and $\frac{\partial w_{sq}}{\partial w_q}$ with an identity mapping, to be $\frac{\partial w_q}{\partial w_s} \approx 1$ and $\frac{\partial w_{sq}}{\partial w_q} \approx 1$ respectively. In other words, we use the identity mapping to simply pass through the gradient untouched to overcome the problem of the gradient of round() and sign() operations being zero almost everywhere.

For activation quantization, we use parameterized clipping technique, PACT [3]. From our experiment, PACT [3] works better than static clipping to [0, 1] based activation quantization methods [37, 39, 21].

Tying quantization methods to sparsity: Depending on quantization methods, there is a case that some weights are quantized to zero giving *free* sparsity. For instance, WRPN [21] quantizes small weights to 0 due to clipping to [-1, 1] with implicit min of 0 and max of 1, while DoReFa-Net [37] is not necessary to map the same small weights to 0, due to prior tanh transformation before quantization. Mainly due to the (disconnected) bi-modal distribution of sparse weights, in Figure 2, we choose to use min-max quantization function to have finer quantization steps only on non-zero elements of sparse weights which in turn reduce quantization error. Noting that our method is not generating additional sparsity since the min value is always greater than zero and gets larger, as sparsity controlling σ becomes large.

3.3.3 Overall Algorithm

The procedure of our proposed SQuantizer is described in Algorithm 1. Similar to [15], we introduce a *Delay* parameter to set a delay for weight SQuantization. In other words, we start quantizing activations but defer SQuantizing weights until *Delay* iterations to let weights stabilize at the start of training, thus converge faster. From our experiments, *Delay* of one third of total training iterations works well across different types of networks, and training from scratch with *Delay* performs better than training without *Delay*. We believe the reason is because training from scratch with *Delay* allows enough time for weights to stabilize and fully adapt the quantized activation.

4. Experiments

To investigate the performance of our proposed SQuantizer, we have conducted experiments on several popular CNNs for classification task using ImageNet dataset, and even extended to object detection task using Pascal VOC.

Algorithm 1 SQuantization for sparse and k-bit quantized neural network

Input: Training data, Weight SQuantization *Delay*, Sparsity controlling σ , Low-bitwidth *k*.

Output: A sparse and k-bit quantized model $M_{SQ}^{\text{sparse}, k}$

```

1: Step 1: Quantize Activation:
2: for iter = 1, ..., Delay do
3:   Randomly sample mini-batch data
4:    $W_{full} \leftarrow W_{full}$ 
5:    $Act_{kbit} \leftarrow Act_{full}$ 
6:   Calculate loss with cross-entropy and weight decay
7:   Update  $W_{full}$ 
8: end for
9: Step 2: SQuantize weights and activations to k-bit:
10: for iter = Delay, ..., T do
11:   Randomly sample mini-batch data
12:    $W_{sparse} \leftarrow W_{full}$  with  $\sigma$ 
13:    $W_{sparse, quantized} \leftarrow W_{sparse}$  with k-bit
14:    $Act_{kbit} \leftarrow Act_{full}$ 
15:   Calculate loss with cross-entropy and weight decay
16:   Update  $W_{full}$ 
17: end for
```

In particular, we explore the following 4 representative networks to cover a range of different architectures: ResNet18 with basic blocks [11], pre-activation ResNet50 with bottleneck blocks [11], MobileNet-v2 [27] with depth-wise separable, inverted residual and linear bottleneck. Furthermore, Darknet-19 for YOLO-v2 object detection [25].

Implementation details: We implemented SQuantizer in PyTorch [24] and used the same hyper-parameters to train different types of networks for ImageNet classification task. During training, we randomly crop 224×224 patches from an image or its horizontal flip, and normalize the input with per-channel mean and std of ImageNet with no additional data augmentation. We use Nesterov SGD with momentum of 0.9 and weight decay of $1e^{-4}$, and learning rate starting from 0.1 and divided by 10 at epochs 30, 60, 85, 100. Batch size of 256 is used with maximum 110 training epochs. For evaluation, we first resize an image to 256×256 and use a single center crop 224×224. Same as almost all prior works [37, 20, 39, 3], we don't compress the first convolutional (conv) and the last fully-connected (FC) layer, unless noted otherwise. It has been observed that pruning or quantizing the first and last layers degrade the accuracy dramatically, but requires relatively less computation.

For object detection task, we use open source PyTorch version of YOLO-v2⁴ with default hyper-parameters as our baseline and apply our SQuantizer on Darknet-19 backbone classifier for YOLO-v2.

⁴<https://github.com/longcw/yolo2-pytorch>

4.1. Evaluation on ImageNet

We train and evaluate our model on ILSVRC2012 [26] image classification dataset (a.k.a. ImageNet), which includes over 1.2M training and 50K validation images with 1,000 classes. We report experiment results of sparse 4-bit quantized ResNet18, ResNet50 and MobileNet-v2, and sparse 2-bit ResNet50, with comparison to prior works. It should be noted that almost all prior low-precision quantization works have not considered efficient network architecture like MobileNet. We believe that, despite the difficulty, our experiment on MobileNet-v2 will provide meaningful insights on the trade-offs between accuracy and model size, especially for severely resource-constrained edge devices.

4.1.1 Sparse and 4-bit quantized Networks

ResNet18: Table 3 shows the comparison of Top1 validation accuracy for 4-bit (both weight and activation) quantized ResNet18 with prior works. Noting that the DoReFa-Net [37] number is cited from PACT [3], and in the case of our *-Quantizer*⁵, we set t_l^n to 0 in Equation 3 to disable sparsification prior to quantization. Our SQuantizer outperforms the prior works, even with 57% of sparsity. Additionally, our *-Quantizer* achieves the state-of-the-art accuracy of 4-bit ResNet18 without sparsification.

Table 3. Comparison of validation accuracy of 4-bit quantized ResNet18 on ImageNet. W and A represent weight and activation respectively.

	DoReFa-Net* [37]	PACT[3]	Our <i>-Quantizer</i>	Our SQuantizer
	Top1	Top1	Top1	Top1 Sparsity (%)
baseline	70.2	70.2	70.4	70.4 -
(4W, 4A)	68.1	69.2	69.7	69.4 57

In Table 4, our *-Quantizer* is used for dense and 4-bit models (4W, -A), while our SQuantizer with σ of 0 is used for sparse and 4-bit quantized models (sparse (4W, -A)). When SQuantizing both weights and activations, we achieve up to 18 \times compression rate within 1% drop in accuracy. Due to the uncompressed last FC layer, the overall sparsity is slightly lower than the conv sparsity (57% vs 60%).

Table 4. SQuantization on ResNet18 on ImageNet

	Accuracy (%)		Sparsity (%)		#Params (M)	Compression Rate
	Top1	Top5	Conv	All		
baseline	70.4	89.7	-	-	11.7	-
(4W, 32A)	70.2	89.4	-	-	11.7	8 \times
sparse (4W, 32A)	69.8	89.2	60	57	5.0	18\times
(4W, 4A)	69.7	89.1	-	-	11.7	8 \times
sparse (4W, 4A)	69.4	89.0	60	57	5.0	18\times

Figure 4 plots Top1 validation accuracy over training epochs for baseline, sparse 4-bit weights, and sparse 4-bit weights with 4-bit activations. The *Delay* in the plot is

⁵Our *-Quantizer* is used to represent the proposed SQuantizer with sparsification disabled.

where step 2 starts in Algorithm 1, and accuracy curves of baseline and sparse (4W, 32A), due to the same precision of activation, are the same before *Delay*. Overall, the accuracy curve of our SQuantizer is similar to the baseline and shows some drop right after *Delay* of 35 epochs, but almost recovers back at the next lowering learning rate epoch (60 epochs).

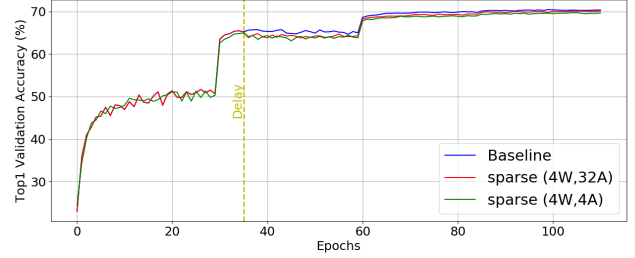


Figure 4. Top1 validation accuracy vs epochs of sparse and 4-bit ResNet18 on ImageNet. Best view in color.

ResNet50: We compare our SQuantizer for 4-bit ResNet50 with many prior works in Table 6. Noting that the DoReFa-Net [37] number is cited from PACT [3] and Apprentice [20] number is directly from an author of the publication. Our SQuantizer outperforms most of the prior works, even with sparsity. However, although PACT [3] shows better accuracy, its baseline accuracy is much higher and still loses 0.4% accuracy after 4-bit quantization. Similarly, we also lose 0.4% accuracy after 4-bit SQuantization, with 41% sparsity (σ of -0.3 is applied).

From Table 5, it is observed that our SQuantizer with 41% sparsity gives almost as accurate as our *-Quantizer*. It indicates that there is an effective value for sparsity controlling σ for each network that allows optimal sparsity with no to little loss in accuracy or even better accuracy due to additional regularization. Automatically finding out the effective value for σ for each network will remain as our future work. In practice, we first try with σ of 0 and then lower or raise σ value according to the training loss in the first few epochs, and estimate the corresponding final sparsity from the intermediate model after step 1 in Algorithm 1.

Table 5. SQuantization on ResNet50 on ImageNet

	Accuracy (%)		Sparsity (%)		#Params (M)	Compression Rate
	Top1	Top5	Conv	All		
baseline	76.3	93.0	-	-	25.5	-
(4W, 32A)	76.4	93.0	-	-	25.5	8 \times
sparse (4W, 32A)	76.0	92.9	60	55	11.5	17\times
(4W, 4A)	76.0	92.7	-	-	25.5	8 \times
sparse (4W, 4A)	75.9	92.7	45	41	15.0	13 \times
sparse (4W, 4A)	75.5	92.5	60	55	11.5	17\times

Table 5 and Table 2 show that within 1% accuracy drop, we can achieve up to 17 \times compression rate for both weight and activation SQuantization, and up to 30 \times compression rate for weights only SQuantization. In addition, our *-Quantizer* for 4-bit weights performs slightly better than the baseline.

Table 6. Comparison of validation accuracy of 4-bit quantized ResNet50 on ImageNet. W and A represent weight and activation respectively.

	DoReFa-Net*[37]		Apprentice*[20]		Low-bit CNN[39]		PACT[3]		Our -Quantizer		Our SQuantizer		
	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Sparsity (%)
baseline	75.6	92.2	76.2	-	75.6	92.2	76.9	93.1	76.3	93.0	76.3	93.0	-
(4W, 4A)	74.5	91.5	75.3	-	75.7	92.0	76.5	93.2	76.0	92.7	75.9	92.7	41

MobileNet-v2: MobileNets are considered to be less attractive to network compression due to its highly efficient architecture, and as such present an ardent challenge to any new network compression technique. To be more specific, MobileNet-v2 uses $3\times$ and $7\times$ smaller number of parameters than ResNet18 and ResNet50 respectively.

Taking up this challenge, we applied our SQuantizer on the latest MobileNet-v2 to further reduce model size and compared with prior work, in Table 7 and Table 8. Different from other networks, we sparsify the last FC layer, since the last FC layer alone uses about 36% of total parameters, while all conv layers consume about 63%, and the rest of 1% is used in batch normalization. Table 7 shows that our SQuantizer outperforms the prior work by large margin, even with sparsity. Additionally, our -Quantizer achieves the state-of-the-art accuracy of 4-bit MobileNet-v2 without sparsification.

Table 7. Comparison of validation accuracy of 4-bit quantized MobileNet-v2 on ImageNet. W and A represent weight and activation respectively.

	QDCN[17]		Our -Quantizer		Our SQuantizer	
	Top1	Top5	Top1	Top5	Top1	Sparsity (%)
baseline	71.9	89.9	72.0	90.4	72.0	-
(4W, 32A)	62.0	89.9	71.2	90.4	70.7	25

Table 8. SQuantization on MobileNet-v2 on ImageNet

	Accuracy (%)		Sparsity (%)			#Params (M)	Compression Rate
	Top1	Top5	Conv	FC	All		
baseline	72.0	90.4	-	-	-	3.6	-
(4W, 32A)	71.2	89.9	-	-	-	3.6	$8\times$
sparse (4W, 32A)	70.7	89.5	21	32	25	3.1	$9\times$
sparse (4W, 32A)	70.5	89.5	30	32	30	2.5	$11\times$
sparse (4W, 32A)	70.2	89.3	35	51	40	2.1	$13\times$
(4W, 4A)	70.8	89.7	-	-	-	3.6	$8\times$
sparse (4W, 4A)	70.3	89.5	21	32	25	3.1	$9\times$

Table 8 shows that within 2% drop in accuracy, we can achieve up to $13\times$ compression rate for weight SQuantization, and up to $9\times$ for both weight and activation SQuantization. It is also observed that when applying our -Quantizer for dense 4-bit weight model, it already lost 0.8% accuracy. For reference, ResNet18 lost 0.2% for the same configuration. As expected, the effect of quantization is more significant on efficient network like MobileNet-v2. However, we believe this trade-offs between accuracy and compression rate can help to determine the applicability of deployment on severely resource-constrained edge devices.

Figure 5 plots Top1 validation accuracy over training epochs for baseline, dense 4-bit weights, and sparse 4-bit weights with various sparsity degree. At *Delay* of 38

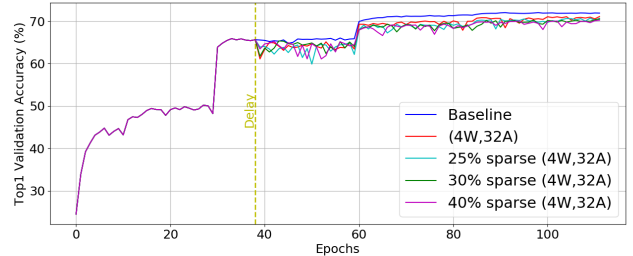


Figure 5. Top1 validation accuracy vs epochs of sparse 4-bit MobileNet-v2 on ImageNet. Best view in color.

epochs, we start SQuantizing weights based on the same baseline with different configurations. It is seen that the accuracy curves of sparse 4-bit models become less stable with higher sparsity. Comparing to the dense 4-bit model, we can have up to 40% sparsity within 1% drop in accuracy. It is worth noting that training with *Delay* allows us to reuse the same intermediate model for experiments with different configurations and shortens training time.

4.1.2 Sparse and 2-bit quantized Network

We SQuantize both weight and activation for 2-bit ResNet50, and compare with prior works in Table 9. The table contains two different comparisons, without and with full-precision shortcut (fpzc). In the case of with fpzc, we don't SQuantize input activations and weights in the shortcut (residual) path, as suggested by PACT-SWAB [2]. Also note that the DoReFa-Net [37] number is cited from Low-bit CNN [39] and Apprentice [20] used 8-bit activation.

For 2-bit quantization, we slightly modified the min and max functions in Equation 3, for better performance. According to Equation 3, all non-zero weights are quantized to either the smallest or largest values of non-zero weights with its own sign. To reduce the significant impact of largest and smallest values, we use mean and std of non-zero elements of sparse weights to determine new values of min and max. To be specific, we use the mean of non-zero weights as a value of new min and the sum of the mean and two std as a value of new max. Then, we clamp the absolute values of sparse weights with these new min and max.

As shown in Table 9, in the case of without fpzc, our SQuantizer outperforms the prior works by large margin, even with 40% sparsity. In the case of with fpzc, our SQuantizer with 36% sparsity gives comparable performance considering that our baseline is (0.6%) lower but gives (0.3%)

Table 9. Comparison of validation accuracy of 2-bit quantized ResNet50 on ImageNet. W and A represent weight and activation respectively. The last two works (PACT-SWAB* [2] and Our SQuantizer*) use full-precision shortcut.

	DoReFa-Net*[37]		Apprentice[20]		Low-bit CNN[39]		PACT[3]		Our SQuantizer			PACT-SWAB*[2]		Our SQuantizer*		
	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Sparsity (%)	Top1	Top5	Top1	Top5	Sparsity (%)
baseline	75.6	92.2	76.2	-	75.6	92.2	76.9	93.1	76.3	93.0	-	76.9	-	76.3	93.0	-
(2W, 2A)	67.3	84.3	72.8(8bit Act)	-	70.0	87.5	72.2	90.5	73.0	91.0	40	74.2(fpsc)	-	73.9(fpsc)	91.6	36

smaller accuracy drop compared to PACT-SWAB [2]. Although we used σ of -0.3 for both the experiments, the sparsity of the latter is lower due to uncompressed shortcut path.

From Table 2, it is already seen that for sparse and 2-bit weight ResNet50, we can achieve up to $42\times$ compression rate within 2% accuracy drop. It is worth noting that our SQuantizer neither demands an auxiliary network to guide [20] nor require two-stage training [19] to achieve state-of-the-art accuracy for sparse and low-precision models.

4.2. Extension to Object Detection

We further applied our SQuantizer to object detection task using YOLO-v2 [25] trained on Pascal VOC. The training dataset of VOC2007 [4] and VOC2012 [5] are used for training, while VOC2007 [4] test dataset is used for evaluating our method. The datasets contains 20 classes and test has been done on input images after re-scaling to 416×416 pixels. We applied our SQuantizer on Darknet-19 which is used as a backbone classifier for YOLO-v2 object detection and conducted two experiments with 4 and 2 bits of precision. Similar to classification task, we didn't compress the first and last conv layer. The σ of 0 and -0.3 are used for 4-bit and 2-bit experiments respectively, and *Delay* of 80 epochs is set for both cases. Same as the baseline set-up, initial learning rate is set to $1e^{-3}$ and divided by 10 at epoch 60 and 90.

Table 10. SQuantization on YOLO-v2 object detection with Darknet-19 on Pascal VOC.

	mAP (%)	Sparsity (%)	#Params (M)	Compression Rate
baseline	70.76	-	67.1	-
sparse (4W, 32A)	69.84	55	30.2	17\times
sparse (2W, 32A)	68.80	40	40.6	26\times

As shown in Table 10, we can achieve $17\times$ compression rate for sparse and 4-bit weights within 1% mAP drop, while we can get $26\times$ compression rate for sparse and 2-bit weights within 2% mAP drop. This proves that our method is promising. Moreover, we show sample output images from the sparse and 2-bit quantized YOLO-v2 in Figure 6 for visual inspection. We believe that this extension to object detection task demonstrates the good general applicability of our proposed SQuantizer.

5. Discussion

Mathematically, for achieving $8\times$ compression rate, we need to either quantize a model with 4-bit precision or sparsify it with at least 87.5% sparsity level to have an equal



Figure 6. Output images from sparse and 2-bit quantized YOLO-v2 with Darknet-19 as a backbone classifier. Best view in color.

rate regardless of the storage overhead of non-zero elements indices. From this calculation, we can infer that low-precision (4-bit or lower) quantization can easily drive higher compression rate than sparsification. However, prior works have shown that 2-bit or lower precision quantization results in significant accuracy degradation. For example, the state-of-the-art accuracy for ResNet50 (before our work) was 72.2% (4.7% drop) from [3], when quantizing both weights and activations with 2-bit precision. Although the high ($16\times$) compression rate is attractive, the degraded accuracy may not be acceptable for real-world applications. This motivated us to design a joint optimization technique for sparsity and quantization, achieving maximal compression while keeping the accuracy close to the original model. As a result, we achieve $17\times$ compression for ResNet50 when SQuantizing both weights and activations with 4-bit precision and 41% sparsity within 1% drop in accuracy. Particularly for resource-constrained edge devices, where both high compression rate and high accuracy are important, this work can enable wider deployment of high-performing deep neural networks on such devices.

6. Conclusion

In this paper, we proposed SQuantizer, a new training method that enables simultaneous learning for sparse and low-precision (weights and activations) neural networks while achieving high accuracy and high compression rate. SQuantizer gives state-of-the-art accuracies using 4-bit and 2-bit precision for ResNet18, ResNet50 and MobileNet-v2, even with high degree of sparsity. Additionally the extension to YOLO-v2 for object detection demonstrates the viability and more general applicability of the proposed SQuantizer to broader vision neural network tasks.

To the best of our knowledge, this paper is the first approach to demonstrating simultaneous learning for both sparse and low-precision neural networks to help the deployment of high-performing neural networks on resource-constrained edge devices for real-world applications.

References

- [1] Y. Bengio. Estimating or propagating gradients through stochastic neurons for conditional computation. *arxiv preprint arxiv:1308.3432*.
- [2] J. Choi, P. I.-J. Chuang, Z. Wang, S. Venkataramani, V. Srinivasan, and K. Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *CoRR*, abs/1807.06964, 2018.
- [3] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.
- [4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [6] A. Gordon, E. Eban, O. Nachum, B. Chen, T.-J. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. *CoRR*, abs/1711.06798, 2017.
- [7] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.
- [8] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.
- [9] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 10 2016.
- [10] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.
- [11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [12] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.
- [13] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.
- [14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [15] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. 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.
- [16] J. W. Jian-Hao Luo and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017.
- [17] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018.
- [18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.
- [19] M. Mathew, K. Desappan, P. K. Swami, and S. Nagori. Sparse, quantized, full frame cnn for low power embedded devices. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 328–336. IEEE, 2017.
- [20] A. Mishra and D. Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *International Conference on Learning Representations*, 2018.
- [21] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr. WRPN: Wide reduced-precision networks. In *International Conference on Learning Representations*, 2018.
- [22] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.
- [23] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. Senn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17*, pages 27–40, New York, NY, USA, 2017. ACM.
- [24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- [25] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.
- [26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [28] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.
- [29] C. Tai, T. Xiao, X. Wang, and W. E. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.
- [30] X. Xu, M. S. Park, and C. Brick. Hybrid pruning: Thinner sparse networks for fast inference on edge devices. *arXiv preprint arXiv:1811.00482*, 2018.

- [31] T. Yang, Y. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6071–6079, July 2017.
- [32] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.
- [33] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76, July 2017.
- [34] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.
- [35] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.
- [36] A. Zhou, A. Yao, K. Wang, and Y. Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9426–9435, 2018.
- [37] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.
- [38] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.
- [39] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid. Towards effective low-bitwidth convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [40] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. 2017.