SeerNet: Predicting Convolutional Neural Network Feature-Map Sparsity through Low-Bit Quantization

Shijie Cao*[1], Lingxiao Ma*[2], Wencong Xiao*[3], Chen Zhang†[4], Yunxin Liu[4], Lintao Zhang[4], Lanshun Nie[1], and Zhi Yang[2]

[1]Harbin Institute of Technology [2]Peking University [3]Beihang University [4]Microsoft Research
{v-shicao,v-lima,v-wencxi,zhac,yunxin.liu,lintaoz}@microsoft.com,
nls@hit.edu.cn,yangzhi@pku.edu.cn

Abstract

In this paper, we present a novel and general method to accelerate convolutional neural network (CNN) inference by taking advantage of feature map sparsity. We experimentally demonstrate that a highly quantized version of the original network is sufficient in predicting the output sparsity accurately, and verify that leveraging such sparsity in inference incurs negligible accuracy drop compared with the original network. To accelerate inference, for each convolution layer, our approach first obtains a binary sparsity mask of the output feature maps by running inference on a quantized version of the original network layer and then conducts a full-precision sparse convolution to find out the precise values of the non-zero outputs. Compared with existing work, our approach avoids the overhead of training additional auxiliary networks, while is still applicable to general CNN networks without being limited to certain application domains.

1. Introduction

Today's breakthrough in artificial intelligence often comes from deep neural networks (DNNs) with very large multi-layer models. Inference on these bulky models often requires a huge amount of computational power and a large amount of energy. Running these models in a low-cost, energy-efficient and low-latency way is highly desirable and has attracted much attention in the research community.

Exploring sparsity in DNNs is a key technique to reduce model-inference cost. Many DNNs, particularly convolutional neural networks (CNNs), are highly sparse with many values involved in the calculation being zero or close to zero. By skipping the computation involving zero values, the model-inference cost could be significantly reduced. Sparsity can arise in several different places in neural network inference. *Weight sparsity* in CNNs has been exten-

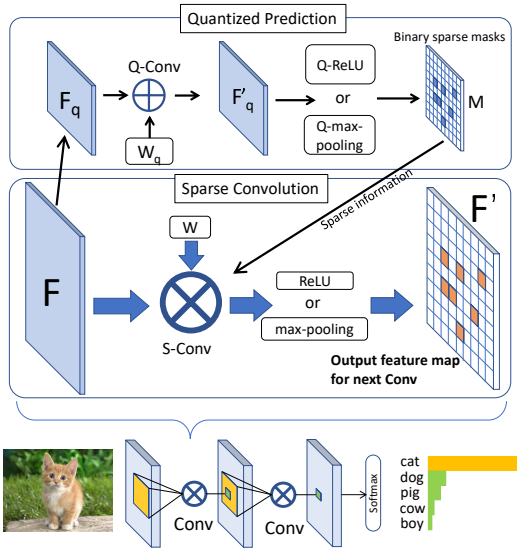


Figure 1. Overall flow of our SeerNet inference for a convolution layer. The sparsity prediction is done layer by layer. The weights of the original layer are denoted as W and the corresponding weights of the quantized layer are denoted as W_q. The input feature map F (or input image for the first CNN layer) is quantized into F_q. Based on W_q and F_q, we execute the quantized low-bit inference (*Q-Conv* and *Q-ReLU*, i.e., quantized convolution and quantized ReLU activation) to generate the sparsity mask M. With M, we execute the full-precision sparse inference (*S-Conv*, i.e., sparse convolution) over W and F to get the output feature map F', which is also the input of the next layer.

*Contribution during internship at Microsoft Research.
†Corresponding author.

sively explored in many previous studies [8, 32, 10, 12, 19]. Some researchers further explore speedup potential of *input sparsity*, such as skipping zeros from ReLU activation [25, 21] or sparse inputs of 3D object classification [33] and detection [34, 26].

Another type of sparsity that can be exploited is *output sparsity*. If some output values are known to be zero, we can avoid computing those outputs altogether. One line of work predicts output feature-map sparsity through external application specific knowledge [23, 14]. These approaches can only work on specific tasks and are not generally applicable to regular CNN networks. Another research direction related to our work is that training a small collaborative network to predict output sparsity [3, 4]. These works have shown meaningful speedup on inference workloads, but they require training of additional neural networks, which is often a daunting task for non-experts.

In this paper, we propose SeerNet, a novel approach to accurately predict output feature-map sparsity of CNN layers. The key idea of SeerNet is first running a highly-quantized (e.g., 4-bit or 1-bit) version of the original CNN network to predict a binary sparsity mask of the output feature maps. We then use this binary sparse mask to guide the full precision convolution, as shown in Figure 1. Since we quantize the original network, our method does not require re-training or integrating external knowledge.

We address two key challenges of the proposed approach. First, the quantized prediction must predict output sparsity accurately while incurring little computation overhead. Second, full precision sparse convolution must be able to efficiently utilize this sparsity to speed up inference. To this end, we develop several techniques for efficient offline network quantization and online quantized inference and propose a fast sparse convolution implementation to take advantage of feature-map sparsity. We have verified our idea and evaluated our system using various popular CNN models over the CIFAR-10 [16] and ILSVRC-2012 [24] datasets. Experimental results demonstrate that our approach is able to predict the feature-map sparsity of the models at an accuracy of 96.5% on average, leading to a negligible drop of the model-inference accuracy of only 0.18% to 0.42%. We also demonstrate apparent wall-clock speedup compared to previous work.

The main contributions of this paper are as follows.

- We propose a novel approach for accurate prediction of CNN feature-map sparsity through low-bit quantization. We develop multiple techniques to ensure good sparsity prediction accuracy and low prediction overhead.
- We provide a system implementation to leverage the predicted feature-map sparsity to accelerate CNN model inference. We employ several optimization approaches to fully leverage the hardware capabilities for practical speedup.
- We conduct comprehensive experiments to demonstrate that our approach achieves speedup in inference with a negligible drop of model accuracy.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 explores the opportunities of feature-map sparsity for accelerating CNN model inference. Section 4 proposes SeerNet to predict feature-map sparsity accurately through low-bit quantization. Section 5 demonstrates CPU speedup by leveraging output feature-map sparsity. Section 6 reports the experimental results and Section 7 concludes this paper.

2. Related Work

Weight pruning. Weight sparsity has been extensively explored in previous work. Since model weights will not change after model training and are constant during inference, previous work proposes sparse matrix algorithms with statistical weight sparsity masks [10, 12, 19, 32, 8, 35] that achieve significant model compression rate and inference speedup. With respect to different granularities of the pruning methods, weight pruning can be applied through fine-grained pruning, filter level pruning, and channel level pruning. Our work focuses on feature-map sparsity and thus is complementary to weight sparsity.

Input sparsity. Researchers have also proposed to take advantage of sparsity in the activation maps [13, 25, 5, 21]. Rectified linear unit (ReLU) activation often contains more than 50% zeros on average. Different from weight pruning, input activation sparsity is dynamically generated during inference. Both hardware-based and software-based convolution algorithms are proposed to exploit input sparsity. However, convolution with input sparsity is hard to be accelerated and may even be slower than dense convolution, due to non-contiguous memory access and worse parallelism. By leveraging output sparsity, our method avoids a lot of computation but keeps a regular memory access pattern, because our inputs are all dense matrices.

Output sparsity. Several methods have been proposed to predict sparsity in output feature maps. X. Dong et al. proposed adding a small auxiliary network for each convolution layer to predict attention areas and skipping computation of those unimportant activation spaces according to the auxiliary network's prediction [3]. In vehicle detection applications, SBNet [23] uses prior knowledge, either from offline maps or online prediction neural networks, to generate computation masks of sparse blocks to speed up inference. M. Figurnov et al. studied how to skip an adaptive number of layers in CNN for unimportant regions in object classification tasks [4]. X. Li et al. proposed to use a pixel-wise mask for re-weighting the computation in the

context of semantic segmentation [18]. These methods are closely related to our work. Compared to them, our method does not require additional model training or prior domain knowledge. Thus, our method can support existing models and applications with minimal efforts from developers. Our method also achieves better prediction accuracy than existing methods. In [1, 28], the authors designed customized accelerators for DNNs to make early decisions or predictions of skipping unnecessary computation.

Quantization. Quantization is a widely-used technique for model compression. V. Vanhoucke et al. demonstrated 8-bit quantization on speech recognition tasks with no quality degradation [31]. With comprehensive re-training strategies, further works quantized the number of bits of convolution kernels from 32 to 8 or even to 4 [39, 36]. Other studies [38, 7, 17, 22] further compressed model size with some weight-sharing techniques. XNOR net [22] quantized AlexNet and VGG with 1-bit weights, but suffered from a significant accuracy loss. Dorefa [37] generalized the quantization method and demonstrated good results with low-bit neural network training. To maintain the model accuracy, retraining is often necessary. In summary, there is a fundamental trade-off between model accuracy and quantization level. Popular practices generally use 16-bit or 8-bit quantization. In this work, we use quantization to predict feature-map sparsity rather than to compress the full model. We show that it is feasible to quantize models more aggressively while still achieving an accurate prediction of feature-map sparsity.

3. Feature-Map Sparsity in CNN

Feature maps in CNN models usually have high sparsity. This is because a convolution layer is commonly followed by a ReLU activation layer that turns all negative inputs into zeros, making the output (i.e., feature maps) of the CNN layer highly sparse. In addition, max-pooling layer only selects a max value in a sub-region and drops other values in the region. As shown in Figure 2, we have observed that the average feature-map sparsity ratios (after ReLU) in widely-used CNN models are between 40% and 80%. Look more deeply, different layers may have different feature-map sparsity ratios. Figure 3 shows the detailed breakdown of the feature maps of layers in VGG16. For layers with additional max-pooling (+MP), sparsity can reach more than 80%, yielding a potential of 5x or more speedup by skipping unnecessary computation of zero outputs.

However, it is challenging to leverage feature-map sparsity because such sparsity heavily depends on CNN inputs and thus cannot be pre-determined without executing the model inference. Thus, we need a method to predict feature-map sparsity. Previous work either trains a collaborative small network [3] or uses external domain-specific knowledge [23] to help predict output sparsity. Both approaches

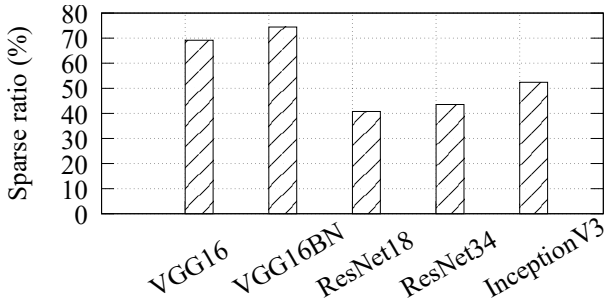


Figure 2. Average sparsity ratios of feature maps after ReLU activation in popular models.

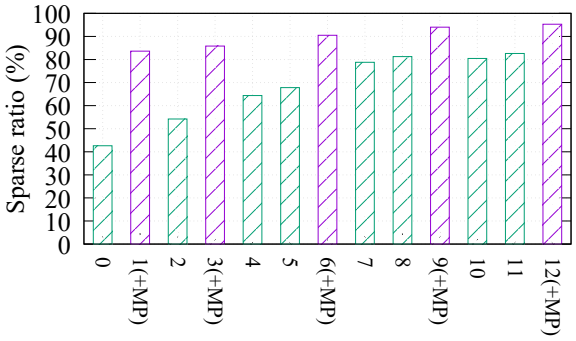


Figure 3. Per-layer feature-map sparsity of VGG16.

are not fully satisfactory. Training a collaborative network is often a formidable task for inexperienced developers, with many additional hyper-parameters to tune, while external domain-specific knowledge is only available on very limited tasks such as semantic segmentation. In Section 4, we describe how our proposed approach can predict feature-map sparsity accurately through low-bit quantization.

4. Predicting Feature-Map Sparsity through Low-Bit Quantization

In SeerNet, we propose to use quantized convolution to predict the feature-map sparsity of CNNs. For a given CNN model, the quantized weights can be generated online or offline. The online computation overhead is negligible because it has high parallelism and low computation complexity. The computation complexity of quantization is only 1/(HW) of the full convolution, where H, W are dimensions of the output feature map. Offline preparation is an option to eliminate online quantization overhead but requires further storage.

During online model inference, we introduce an extra "sparsity prediction" step for each CNN layer. We first use the quantized weights to perform quantized convolution over the input data and generate a binary sparsity mask. Using the sparsity mask, we then use the original CNN weights to conduct sparse convolution and obtain the output feature map, which is also the input to the next layer. Sparsity pre-

diction and followed sparse convolution are done in a layer-by-layer manner. Figure 1 shows the overall flow of the quantized feature-map sparsity prediction and sparse inference for a single CNN layer.

There are two critical requirements for our feature-map sparsity prediction in SeerNet: 1) it must ensure the sparsity prediction accuracy and final model accuracy, and 2) the prediction process must be fast and incur low computational overhead. To meet these requirements, we develop three techniques, namely *efficient quantizer*, *dequantization-free integer convolution*, and *quantized sparsity-mask prediction*. In the following subsections, we describe how each technique works in detail.

4.1. Efficient Quantizer

Quantization is a popular method to accelerate neural network inference and training. Different from classical quantization, we do not use it for full inference but only use it in a layer-by-layer manner to predict output feature-map sparsity. Therefore, we can use *much lower* bits than those used in quantization schemes for carrying out full models. For the output of ReLU activation, the prediction needs to find the signs of output feature maps and zero out those negative ones. For max-pooling, this prediction needs to find the position of the largest value in a sub-region with no regard to the precise values. Lower quantization bits often mean lower computation overhead and faster inference speed. Still, over quantization introduces too much prediction errors and will degrade model accuracy. We determine the optimal quantization level empirically. We show experiments to find the lowest quantization level in Section 6.5.

Many quantization methods are proposed before. We use a popular method similar to the quantization method used in TensorFlow [15] due to its efficiency and high precision. Figure 4 shows a toy example of 4-bit quantization to a 3x3 tensor. We first find the max absolute value of the tensor, which is '1.2' in this case. We then define a linear mapping function to map the largest value '1.2' to the maximum integer representation, which is $2^{4-1} - 1 = 7$ in this case. So, any number between $-1.2\ to\ 1.2$ is linearly mapped to $-8\ to + 7$ by the mapping function $y = int(\frac{x-(-1.2)}{1.2-(-1.2)} \times 8)$.

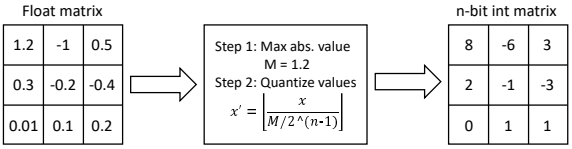


Figure 4. An example of n-bit quantization (e.g., n=4).

4.2. Dequantization-free Integer Convolution

Here we introduce our quantized convolution operator (Q-Conv). Unlike traditional methods, our quantized convolution does not need the de-quantization stage to recover precision and thus needs fewer operations and runs faster. Equation 1 denotes a classical convolution.

$$Y = \sum_{i}^{N} W_i \bigotimes X_i \tag{1}$$

where $\otimes$ denotes a convolution operation. We ignore bias for the sake of simplicity. Given a quantizer f, a quantized convolution computation is shown in equation 2, where $\oplus$ denotes the integer convolution operations.

$$\begin{aligned} f(Y) &= f(\sum_{i}^{N} W_i \bigotimes X_i) \\ &= \sum_{i}^{N} f(W_i \bigotimes X_i) \\ &= \sum_{i}^{N} f_{w\times x}^{-1}(f_w(W_i) \bigoplus f_x(X_i)) \end{aligned} \tag{2}$$

Different from classical quantized convolution, our Q-Conv is dequantization-free because we only care about the signs for ReLU and the max-value position for max-pooling. Thus, the computation formula is shown as in equation 3.

$$\begin{aligned} sign(f(Y)) &= sign(\sum_{i}^{N} f_{w\times x}^{-1}(f_w(W_i) \bigoplus f_x(X_i))) \\ &= sign(\sum_{i}^{N} (f_w(W_i) \bigoplus f_x(X_i))) \end{aligned} \tag{3}$$

4.3. Quantized Sparsity-Mask Prediction

In many popular CNN models, a convolution layer is often followed by a batch normalization layer or/and a ReLU layer or/and a max-pooling layer. ReLU leads to zero elements and max-pooling discards more unused elements. Using a quantized network to predict the feature-map sparsity after ReLU+max-pooling, we can save more unnecessary computation compared to only predict the feature-map sparsity after ReLU. Different models have different combinations of these layers, depending on how the models are designed and tuned. Specifically for our sparsity-mask prediction, we divide all combinations into two groups.

Convolution + Relu or/and max-pooling. As discussed above, our Q-Conv outputs low-bit integer numbers. When a ReLU layer follows a convolution, we apply a quantized ReLU operation (Q-ReLU) on the output of Q-Conv. Q-ReLU only cares about the signs of Q-Conv's output feature maps and thereby generates a corresponding sparsity mask with the same dimension. Similarly, Q-max-pooling only cares about the position of max value in sub-region and generates a corresponding mask, as is shown in Figure 5.

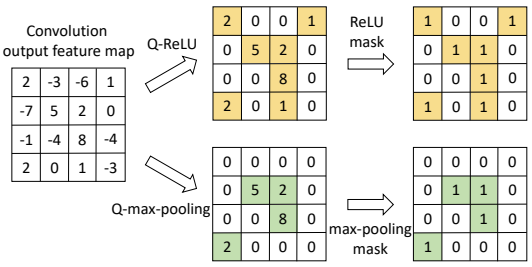


Figure 5. An example of Q-Relu and Q-max-pooling.

Convolution + Batch norm + ReLU/max-pooling. Batch normalization [11] can be applied to reducing feature maps internal co-variant shift and is frequently used in CNN models. From the perspective of arithmetic computation, batch normalization layer has five kinds of parameters, which are scaling factor α , bias β , average mean μ, average variance σ^2, and a small constant included for numerical stability ϵ, as is shown in equation 4.

$$B = \frac{\alpha \times (Y - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta \tag{4}$$

Directly applying quantized batch-normalization (Q-BN) on the output of Q-Conv will result in the product of quantized parameters of two successive layers. This will amplify the precision loss included by quantization and produce extra error in sparsity prediction. As shown in equation 5, parameters of both convolution layer and bath normalization layer need to be quantized, mainly the convolution weights W and batch normalization scaling factor α.

$$B = \frac{\alpha \times (\sum_i^N W_i \bigotimes Xi + bias - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta \tag{5}$$

We remove the compound quantization errors by fusing Q-Conv kernel and Q-BN kernel. Kernel fusion is a common practice for accelerating DNN models. Here we fuse the quantization of convolution and batch normalization to remove compound quantization errors from the cascading layers. Equation 6 shows the deduction of our fused operator, where we fuse α and W_i as $f(\alpha \times W_i)$. We refer to the fused Q-Conv and Q-BN operator as Q-Conv-BN.

$$\begin{aligned} f(B) &= f(\frac{\sum_i^N \alpha W_i \bigotimes X_i + \alpha \times (bias - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta) \\ &= \frac{f(\sum_i^N \alpha W_i \bigotimes X_i) + f(\alpha \times (bias - \mu))}{f(\sqrt{\sigma^2 + \varepsilon})} + f(\beta) \\ &= \frac{\sum_i^N f_w(\alpha W_i) \bigoplus f_x(X_i) + f(\alpha \times (bias - \mu))}{f(\sqrt{\sigma^2 + \varepsilon})} \\ &\quad + f(\beta) \end{aligned} \tag{6}$$

With these techniques, including the efficient quantizer, dequantize-free integer convolution, efficient Q-ReLu and Q-max-pooling kernel, the Q-Conv-BN fusion technique, our sparsity-mask prediction is both fast and accurate.

5. Accelerating CNN Model Inference

In this section, we present our efforts on turning feature-map sparsity into speedup on CPU. Theoretically, given a ReLu layer with 80% sparsity, the upper bound speedup is 5x by skipping 80% computation. For max-pooling layers, a 2x2 max-pooling can save three-quarters computation, which means theoretically a 4x speedup. However, in practice, it is hard to achieve such speedup due to quantized prediction cost and sparse computation overhead. While this work mainly focuses on feature-map sparsity prediction, we develop several techniques to accelerate our quantized sparsity prediction and sparse convolution on commodity hardware.

AVX acceleration. Current commodity-off-the-shelf CPUs do not have native low-bit arithmetic hardware support. Therefore, we take advantage of CPU's vector processing units, such as AVX, to perform quantized prediction. AVX2, or advanced vector extension V2, is an arithmetic hardware in Intel CPUs for vector operations of up to 256-bits. Since current Intel CPUs do not have native support for 4-bit data, we use 8-bit integers for arithmetic computation even if we use a lower bit precision (such as 4bits) for our prediction network. AVX can process 32 8-bit integer operations per cycle in parallel. For efficient storage, we use 4-bit format to cache our intermediate results.

Efficient sparsity-mask encoding format. A good sparse encoding format directly increases sparse convolution's computation efficiency. We propose an efficient encoding format, as shown in Figure 6. In this encoding format, we discard all the indices of zero outputs and thus the S-CONV kernel only takes non-zero entries. In addition, we directly encode matrix indexes so that S-Conv can retrieve indices and input vectors with negligible overhead.

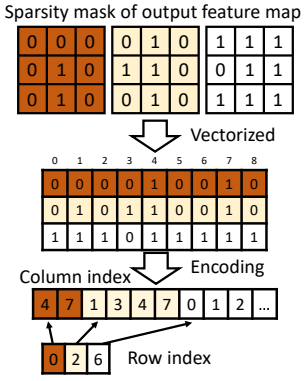


Figure 6. Efficient sparsity-mask encoding.

Multi-level data reuse. Our dual-model inference introduces duplicated storage for quantized parameters and feature maps. In general, additional storage overhead leads to wasteful memory access and therefore downgrades performance. We utilize multi-level data reuse technique to solve this issue. To increase data reuse, we fuse the execution of input quantization and Q-Conv, Q-BN, Q-ReLU, and Q-max-pooling to fully reusing data in CPU registers. Since Q-Conv is packed in AVX vector format, we use in-register arithmetic left and right shift to transform the data format between AVX and other kernel's usage. We also fuse S-Conv with S-BN/S-ReLU/S-max-pooling to reuse temporary data in register and CPU cache.

6. Evaluation

In this section, we conduct experiments on two benchmark datasets with various CNN models. Section 6.1 describes the datasets and models we use. Our evaluation centers around two aspects: the model accuracy of SeerNet (Section 6.2) and the inference speedup of SeerNet achieved on CPU (Section 6.3). Section 6.4, Section 6.5 and Section 6.6 further discuss the quantized sparsity-mask prediction accuracy of each layer, the accuracy sensitivity to quantization bits used for prediction, and the effect of fusing CONV and BN on preserving model accuracy, respectively.

6.1. Datasets and CNN Models

We use two datasets, CIFAR-10 [16] and ILSVRC-2012 [24], to evaluate our proposed method. CIFAR-10 is a dataset for image classification with 60K photographs, which are sized of 32x32 and classified into 10 categories. ILSVRC-2012 is a dataset with 1.28 million images which are classified into 1000 classes.

To evaluate the accuracy of SeerNet and the speedup of convolution with ReLU and max-pooling, we use five representative CNN models. These models include VGG16, VGG16_BN, ResNet18, ResNet34 and Inception [27, 9, 29, 30]. We use pre-trained models on ILSVRC-2012 from Pytorch torchvision model zoo and follow the original data augmentation strategies. For CNN models on CIFAR-10, we train the baseline models using Pytorch.

6.2. Overall Model Accuracy

We evaluate the overall model accuracy of SeerNet by applying sparse convolution with quantized prediction for all layers of the five CNN models *without* re-training on the two datasets. The quantization bit is configured to 4-bit. The compared baselines are pre-trained models with original configurations.

ILSVRC-2012 Table 1 shows the top-1 and top-5 model accuracy on the ILSVRC-2012 dataset. Overall, SeerNet only has a very small accuracy drop with an average value of 0.51% on top-1 and 0.28% on top-5, compared to the baseline among all the five models. SeerNet achieves the minimal accuracy drop on VGG16 of 0.28% and 0.1% for top-1 and top-5, respectively. Although the accuracy drop becomes higher for more complex models, SeerNet also achieves a reasonable result on the InceptionV3 model, i.e., a drop of 0.96% on top-1 and 0.65% on top-5, respectively. Thanks to our Q-Conv-BN fusion technique, batch normalization only has a little effect on model accuracy with a further accuracy drop of 0.24% on top-1 and 0.22% on top-5, respectively, in comparing VGG16 with VGG16_BN. We will further evaluate the effect of batch normalization in Section 6.6.

Model	Baseline (Top1/Top5)	SeerNet (Top1/Top5)	Acc. Drop (Top1/Top5)
VGG16	71.59/90.38	71.31/90.28	0.28/0.10
VGG16_BN	73.37/91.50	72.85/91.18	0.52/0.32
ResNet18	69.76/89.08	69.34/88.90	0.42/0.18
ResNet34	73.30/91.42	72.95/91.25	0.35/0.17
InceptionV3	77.35/93.62	76.39/92.97	0.96/0.65

Table 1. Top-1 and Top-5 accuracy (%) of SeerNet with 4-bit quantized prediction on ILSVRC-2012.

Model	Baseline	SeerNet	Acc. Drop
VGG16	92.57	92.48	0.09
VGG16_BN	93.89	93.60	0.29
ResNet18	93.91	93.88	0.02
ResNet34	94.80	94.76	0.04
InceptionV1	95.12	93.82	1.30

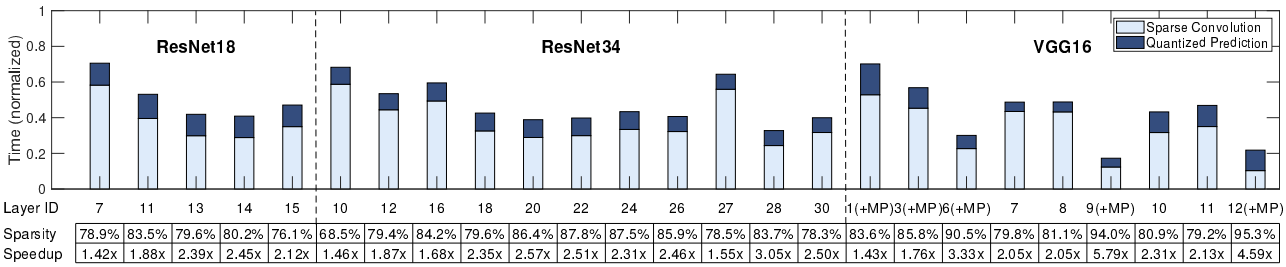
Table 2. Top-1 accuracy (%) of SeerNet with 4-bit quantized prediction on CIFAR-10.

CIFAR-10 Table 2 shows the overall top-1 model accuracy on the CIFAR-10 dataset. SeerNet achieves an average 0.35% accuracy drop among the five models, where the minimal one is only 0.02% on ResNet18 and the maximal one is 1.30% on InceptionV1, respectively. Similar to the results on the ILSVRC-2012 dataset, batch normalization affects little on the quantized prediction. The further accuracy drop is 0.2% when comparing the VGG16 model with batch normalization enabled and disabled.

The evaluation results on both ILSVRC-2012 and CIFAR-10 datasets demonstrate the high accuracy performance of SeerNet. According to Section 4, SeerNet runs sparse convolution based on the quantized sparsity-mask prediction, so the overall model accuracy is only affected by the quantized prediction in SeerNet. We further evaluate the quantized prediction with two micro-benchmarks in Section 6.4 and 6.5.

6.3. Inference Speedup

To demonstrate the achievable speedup with our sparse convolution algorithm, we implement quantized prediction



Layer ID	7	11	13	14	15	10	12	16	18	20	22	24	26	27	28	30	1(+MP)	3(+MP)	6(+MP)	7	8	9(+MP)	10	11	12(+MP)
Sparsity	78.9%	83.5%	79.6%	80.2%	76.1%	68.5%	79.4%	84.2%	79.6%	86.4%	87.8%	87.5%	85.9%	78.5%	83.7%	78.3%	83.6%	85.8%	90.5%	79.8%	81.1%	94.0%	80.9%	79.2%	95.3%
Speedup	1.42x	1.88x	2.39x	2.45x	2.12x	1.46x	1.87x	1.68x	2.35x	2.57x	2.51x	2.31x	2.46x	1.55x	3.05x	2.50x	1.43x	1.76x	3.33x	2.05x	2.05x	5.79x	2.31x	2.13x	4.59x

Figure 7. Inference time and speedup. The total computation time of the SeerNet is summed up by the computation time spent on sparse convolution and quantized prediction. So the bars are the smaller the better. The speedup is reciprocal to computation time.

and sparse convolution kernel by writing C++ code. According to Section 3, layers with higher sparsity usually mean more room for speedup. Due to computation overhead introduced by quantized prediction, layers with small sparsity may not have any speedup. Thus, we only apply sparse convolution on the layers whose sparsity is larger than 60%. For the rest of layers, we use original dense convolution. We test our performance on real image data from ILSVRC2012 dataset.

To have a fair comparison with previous work, we use ResNet18, ResNet34 and VGG16 tested on ILSVRC-2012 dataset. We compare with the following methods. LCCL [3] and PFEC [17] focus on leveraging convolution sparsity for inference speedup. LCCL re-trains a small collaborative network to predict output sparsity. PFEC prunes convolution filters. BWN and XNOR [22] accelerate inference by model quantization. BWN uses binarized convolution weights. XNOR re-trains a network with both binarized convolution input and binarized weights. All above methods demonstrate speedup on CPU. We use single thread OpenBLAS library running on an Intel Xeon CPU E5-2630 v3 (2.40GHz) for comparison with them.

Table 3 shows the wall-clock speedup running on the Intel CPU. We use the performance numbers of LCCL, PFEC, BWN, and XNOR reported in their papers. We achieve an apparent speedup with negligible accuracy loss. Compared with LCCL that is mostly related to our work, our method achieves a smaller accuracy drop (0.42%/0.18% vs. 3.65%/2.30% in ResNet18) and a higher speedup (30.0% vs. 20.5% in ResNet18), without requiring any model re-training effort.

Figure 7 shows detailed layer-wise computation time as well as the corresponding speedup of our sparse convolution implementation compared to OpenBLAS based dense convolution. In this figure, dense convolution's computation time is normalized to 1. The total computation time is summed up by computation time spent on quantized prediction and sparse convolution respectively, as is shown as light blue bars and dark blue bars in this figure. From this figure, we can see more than half of the layers in ResNet18 and ResNet34 can achieve 1.2x to 3.4x speedup. VGG16's convolution layer #3, #6, #9 and #12 are followed by 2x2 max-pooling besides ReLU. Since 2x2 max-pooling select one from four neighbor pixels, it adds more sparsity to output feature maps. So it achieves much higher speedup (up to 5.79x) than those with only ReLU followed.

Model	Method	Top-1 Acc. Drop(%)	Top-5 Acc. Drop(%)	Speedup	Re-train?
ResNet 18	**SeerNet**	**0.42**	**0.18**	30.0%	**No**
	LCCL[3]	3.65	2.30	20.5%	Yes
	BWN[22]	8.50	6.20	50.0%	Yes
	XNOR[22]	18.10	16.00	**98.3%**	Yes
ResNet 34	**SeerNet**	**0.35**	**0.17**	22.2%	**No**
	LCCL[3]	0.43	**0.17**	18.1%	Yes
	PFEC[17]	1.06	-	**24.2%**	Yes
VGG 16	**SeerNet**	**0.28**	**0.10**	**40.1%**	**No**
	PFEC[17]	-	0.15	34.0%	Yes

Table 3. Comparison with previous acceleration work.

Discussion. The predicted output sparsity shows a high theoretical speedup. We use a CPU implementation to demonstrate the efficiency of our idea. Sparse DNN acceleration with specialized accelerators and mixed-precision hardware platform are active research topics which can further speed up our scenario. Nvidia's latest generation Turing GPU integrates 4-bit/8-bit/16-bit mixed precision tensor cores [20], which can further decrease quantized prediction overhead in SeerNet. Our sparse convolution can also be accelerated by leveraging GPU's massive parallel processing units with corresponding optimization techniques. Specialized accelerators on FPGA or ASIC show great potential in bringing a much higher speedup and power efficiency for sparse DNNs [6, 2, 21, 1, 28]. Furthermore, they can provide circuit level bit-manipulation which is suitable to conduct low-bit quantized prediction.

6.4. Quantized Prediction Accuracy

Achieving sufficiently low error rate for feature-map sparsity prediction is important to maintain overall model accuracy. For each input picture's inference, we make a layer-by-layer comparison between quantized and original convolution's output feature maps. Then we count the number of correct predictions by differentiating each pixel's sign

ReLU Layer#	1	2	3	4	5	6	7	8	9	10	11	12	13
Prediction Error Rate	4.3%	9.5%	7.0%	4.8%	4.9%	4.1%	2.1%	2.4%	2.2%	1.0%	2.0%	1.7%	0.7%

Table 4. Feature-map sparsity prediction error rate of VGG16 on ILSVRC-2012 dataset, layer by layer.

of quantized convolution's output feature map with that of the original convolution's output. The quantization bit is set to 4-bit.

Due to the space limitation, Table 4 only shows the quantized prediction error rate of each layer of the VGG16 model on the ILSVRC-2012 dataset. SeerNet can achieve an average error rate of 3.58% on all the layers of the VGG16 model. In detail, the maximal error rate is 9.5% on the 2nd layer and the minimal one is 0.7% on the last layer, respectively.

6.5. Sensitivity Study of Quantization Bits

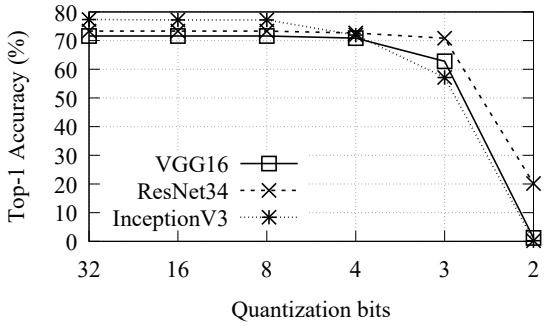


Figure 8. Top-1 accuracy of VGG16, ResNet34 and InceptionV3 with different quantization bits on ILSVRC-2012.

Recall that the only module in SeerNet that affects model accuracy is the quantized prediction. In order to study how the quantization bit affects quantization prediction, i.e., the sensitivity, we experiment on different quantization bits from 32-bit to 2-bit for both input feature maps and convolution weights. In this experiment, we evaluate the top-1 accuracy of the VGG16, ResNet34 and InceptionV3 models running on the ILSVRC-2012 dataset.

Figure 8 reports the results. It is clear that all the three models can achieve a good top-1 accuracy when the quantization bit is more than 4 and the model accuracy drops drastically when we use 3-bit quantization or lower. This is because higher quantization bits can keep more information to make the quantized prediction more precise. Specifically, the accuracy of the VGG16 and the InceptionV3 models will drop close to 0 for 2-bit quantization. To sum up, quantizing too many bits in the quantized prediction will hurt overall model accuracy and 4-bit quantization is a reasonable configuration that can keep enough information for quantized prediction.

6.6. Effect of Fusing Conv and BN

As introduced in Section 4.3, we fuse the CONV layer and its subsequent BN layer in order to eliminate the impact of quantizing two layers continuously on sparsity prediction accuracy. In this section, we conduct an experiment on ILSVRC-2012 with VGG16_BN to demonstrate fusing CONV and BN (Fused_Conv_BN) can increase the overall model accuracy of SeerNet.

Bits	Quant. Fused_Conv_BN	separately Quant. Conv and BN	Acc. Diff.
16-bit	73.36/91.50	73.36/91.50	0.00/0.00
8-bit	73.36/91.50	73.36/91.49	0.00/0.01
4-bit	72.85/91.18	66.70/87.29	6.15/3.89
2-bit	8.68/20.36	0.06/0.59	8.62/19.77

Table 5. SeerNet accuracy (%) comparison between quantizing Fused_Conv_BN and quantizing Conv and BN separately using VGG16_BN on ILSVRC-2012.

Table 5 shows a direct comparison between quantizing Fused_Conv_BN and quantizing Conv and BN separately. With 16-bit and 8-bit quantization prediction in SeerNet, quantizing Fused_Conv_BN and quantizing Conv and BN separately can both preserve the model accuracy. However, with 4-bit quantization, quantizing Fused_Conv_BN has a small impact on model accuracy while quantizing Conv and BN separately decreases model accuracy significantly. Therefore, quantizing Fused_Conv_BN has a lower impact on model accuracy than quantizing Conv and BN separately. This is because quantizing twice accumulates the information loss and increases the sensitivity of model accuracy to quantization bits.

7. Conclusion and Future Work

In this work, we propose SeerNet that accelerates CNN inference by taking advantage of output feature-map sparsity. We propose a novel output sparsity-mask prediction scheme by using highly quantized convolution. We develop multiple techniques to ensure high prediction accuracy as well as extremely low computational overheads. Verified on five popular CNN models and two datasets, we demonstrate an end-to-end speedup on CPU with negligible loss of model accuracy. In the future, we plan to extend this work to hardware platforms that have better mixed precision arithmetic support and massive parallel processing units, like FPGAs and GPUs. Doing so may further reduce the prediction overhead and provide more promising speedup.

References

[1] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K Gupta, and Hadi Esmaeilzadeh. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 662–673. IEEE, 2018.

[2] Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. Efficient and effective sparse lstm on fpga with bank-balanced sparsity. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 63–72. ACM, 2019.

[3] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

[4] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048, 2017.

[5] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[6] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84. ACM, 2017.

[7] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[8] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[10] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2017.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[13] Patrick Judd, Alberto Delmas, Sayeh Sharify, and Andreas Moshovos. Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing. *arXiv preprint arXiv:1705.00125*, 2017.

[14] Tao Kong, Fuchun Sun, Anbang Yao, Huaping Liu, Ming Lu, and Yurong Chen. Ron: Reverse connection with objectness prior networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5936–5944, 2017.

[15] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[18] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3193–3202, 2017.

[19] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[20] Nvidia. Nvidia turing gpu architecture, 2018. `https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf`.

[21] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40. IEEE, 2017.

[22] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[23] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8711–8720, 2018.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Shaohuai Shi and Xiaowen Chu. Speeding up convolutional neural networks by exploiting the sparsity of rectifier units. *arXiv preprint arXiv:1704.07724*, 2017.

[26] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *arXiv preprint arXiv:1812.04244*, 2018.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[28] Mingcong Song, Jiechen Zhao, Yang Hu, Jiaqi Zhang, and Tao Li. Prediction based execution on deep neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 752–763. IEEE, 2018.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[31] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[32] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[33] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[34] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[35] Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. *arXiv preprint arXiv:1811.00206*, 2018.

[36] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.

[37] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[38] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[39] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7920–7928, 2018.