Downscaling and Overflow-aware Model Compression for Efficient Vision Processors

Haokun Li
South China University of Technology
Peng Cheng Laboratory
selihaokun@mail.scut.edu.cn

Jing Liu
Faculty of Information Technology
Monash University
jing.liu1@monash.edu

Liancheng Jia
Peking University
jlc@pku.edu.cn

Yun Liang†
Peking University
Peng Cheng Laboratory
ericlyun@pku.edu.cn

Yaowei Wang†
Peng Cheng Laboratory
wangyw@pcl.ac.cn

Mingkui Tan†
South China University of Technology
Peng Cheng Laboratory
mingkuitan@scut.edu.cn

***Abstract*—Network pruning and quantization are two effective ways for model compression. However, existing model compression methods seldom take hardware into consideration, resulting in compressed models that still take high energy and chip area cost on a vision processor. To address this issue, one may reduce the bit-widths of the accumulator and the multiplier in fixed-point inference to significantly reduce the energy and chip area. However, the numerical error brought from the low-bit multiplier in the downscaling procedure is large, while the low-bit accumulator suffers from the overflow issue. Both of them lead to significant performance degradation. In this paper, we propose downscaling and overflow-aware model compression for efficient vision processors. Specifically, we propose downscaling-aware training to simulate the downscaling procedure during training so that the models are adjusted to inference with low bit-width multipliers. To address the overflow issue, we apply overflow-aware training to gradually reduce the range of quantized values. We further restrict the channel's number of each layer to be the multiple of some value (*e.g.*, 16) to take advantage of parallel computing by channel pruning. With the proposed method, we are able to obtain the compressed model with low bit-width accumulators and multipliers during inference while maintaining the performance. As a result, the energy and chip area cost can be significantly reduced. To demonstrate this, we further co-design an agilely customizable vision processor and its SoC. Extensive experiments on image classification, object detection, and semantic segmentation demonstrate the effectiveness of our proposed method. For example, on ImageNet, our compressed 8-bit ResNet-50 achieves lossless performance with 16-bit accumulators and 12-bit multipliers.**

***Index Terms*—Downscaling-aware; Overflow-aware; Model Compression; Efficient Vision Processor**

I. Introduction

Deep neural networks (DNNs) have achieved tremendous success in many challenging vision tasks, such as image classification [1]–[3], object detection [4], [5], and semantic segmentation [6]. Despite the promising performance, DNNs are extremely difficult to be deployed into edge-AI devices (*e.g.*, vision processor) with limited computational resources due to their massive number of parameters and high computational overhead.

†Co-corresponding author.

TABLE I
Energy and Area cost for floating-point and fixed-point operations (on 45nm 0.9V CMOS) [8]–[10].

Operation	Energy (pJ)	Area (μm^2)
32-bit Floating-point Add	0.9	4184
32-bit Fixed-point Add	0.1	137
8-bit Fixed-point Add	0.03	36
32-bit Floating-point Mult	3.7	7700
32-bit Fixed-point Mult	3.1	3495
8-bit Fixed-point Mult	0.2	282

To reduce the model size and the computational overhead, network quantization is an effective way that maps the full-precision weights and activations into the low-precision ones. In this case, the floating-point operations can be replaced by the fixed-point operations, which significantly reduces the energy and area cost (See Tab. I). For example, we can replace a 32-bit convolution operation with an 8-bit convolution operation. Specifically, we convolve 8-bit weight and 8-bit input using multiply-accumulate (MAC) operation, and store the MAC result in a 32-bit accumulator. Then, we use a 32-bit multiplier to downscale the 32-bit convolution result to the 8-bit one and feed it into the next layer [7]. In this case, the downscaling procedure can be implemented as a fixed-point multiplication with a 32-bit integer (called multiplier) and a bit-shift operation. Due to the high bit-widths of the accumulator and multiplier, it still consumes a lot of energy and chip area cost for the existing vision processors.

In order to reduce the chip area and energy cost, an intuitive method is to use a lower bitwidth multiplier in the downscaling procedure and use a lower bit-width accumulator to store the MAC results. However, the above methods suffer the following limitations. First, when using low bit-width multipliers (*e.g.*, 12-bit), the numerical error brought from each layer's downscaling procedure will be large, leading to significant performance degradation. Second, when using a low bit-width accumulator (*e.g.*, 16-bit), numerical overflow on MAC results becomes a frequently-happening problem that must be explicitly considered [11]. To further reduce the energy cost, we can further improve the computing efficiency by convoluting a group of channels in parallel.

To address the above limitations, we propose a downscaling and overflow-aware model compression scheme for efficient vision processors. Specifically, on the one hand, the proposed downscaling-aware training simulates the downscaling procedure of the convolution layer in the forward pass of training. In this way, the models are adjusted to perform with low bit-width multipliers. On the other hand, the overflow-aware training adaptively determines the quantized value range of the activations and parameters in each layer. In this way, the proposed method is able to prohibit numerical overflow with low bit-width accumulators as much as possible. To take full advantage of parallel computing, we further use channel pruning methods to restrict the number of channels of each layer to be the multiple of some value (*e.g.*, 16). With the proposed method, we obtain the compressed model with low bit-width accumulators and multipliers during inference while maintaining the performance. To demonstrate this, we co-design an agilely customizable vision processor and its SoC with low bit-width accumulators and multipliers.

Our main contributions are summarized as follows:

- We propose a downscaling and overflow-aware model compression scheme for efficient vision processors. With the proposed method, the model can be compressed with little performance degradation when using low bit-width accumulators and multipliers during inference.
- We co-design an agilely customizable vision processor and its SoC with low bit-width accumulators and multipliers, which save the chip area significantly.
- We evaluate our model compression scheme on image classification, object detection, and semantic segmentation over various network architectures. Extensive experiments show the superior performance and efficiency of the proposed method. For example, on ImageNet, our compressed 8-bit ResNet-50 achieves lossless performance with 16-bit accumulators and 12-bit multipliers, reducing 51.44% chip area of the downscaling unit.

II. Related Work

Network Quantization. To reduce the model size and the computational overhead, many network quantization methods [7], [11]–[13] have been proposed to map the full-precision values to low-precision ones (*e.g.*, 8-bit), which is an effective way to improve the inference latency and energy efficiency. It has been shown that quantizing the network's weights and activations to 8-bit is able to obtain a compressed network with small model size and lower computational overhead [7]. However, this method ignores the numerical error brought from each layer's downscaling procedure. In this case, when the bit-width of the multiplier is low (*e.g.*, 12-bit), the numerical error is so large that it leads to significant performance degradation. As for accumulator, an overflow-aware training method [11] is proposed to reduce the numerical overflow when using a low bit-width accumulator. However, numerical overflow still occurs occasionally in this method, which is unacceptable in practical applications.

Algorithm 1 Pipeline of proposed model compression scheme.

```
Input: Pre-trained full-precision model $M$, training data set $\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the bit-width of the accumulator $B_a$, the bit-width of the multiplier $B_m$, the bit-width of activations and weights $k$.
Output: Trained quantized model $P_q$.
1: Prune and finetune the full-precision model $M$ until converge to get the pruned model $P$.
2: Initialize quantized model $P_q$ using $P$.
3: Train $P_q$ via overflow-aware training until converge.
4: Finetune $P_q$ via downscaling and overflow-aware training until converge.
```

Channel Pruning. Channel pruning is a kind of structure pruning, which aims at removing redundant channels to accelerate the run-time inference [14]. Channel pruning is friendly to edge-AI devices because the devices do not need some special implementation to deploy the pruned model. The resulting pruned models take less storage and lower computational overhead. Network Slimming [15] is a classic channel pruning method that automatically identifies unimportant channels and then prunes them. Another channel pruning method DCP [16] proposed a discrimination-aware channel pruning scheme to compress deep models with the introduction of additional discrimination-aware losses. However, DCP and many other methods only prune the middle layer in the bottleneck structure of residual blocks, generating an hourglass structure. To address this, CURL [17] has been proposed to prune channels both inside and outside the residual connection, generating a shape similar to an opened wallet.

III. Downscaling and Overflow-aware Model Compression

In this paper, we propose a downscaling and overflow-aware model compression scheme, which aims to compress the model so that it can be deployed on the processors with lower bit-width accumulators and multipliers. Given a pre-trained model, we first conduct channel pruning to take advantage of parallel computation. Then, we quantize the model and perform overflow-aware training. Last, we conduct downscaling and overflow-aware training to finetune the model. The overall training method is summarized in Algorithm 1.

A. Preliminary: Quantized Training and Inference

Symmetrical and uniform quantization maps a real number r to a finite set of integer values as

$$Q(r, s) = \text{round}\left(\frac{r}{s}\right), \tag{1}$$

where $\text{round}(\cdot)$ returns the nearest integer of a given value and s denotes the real value step size. Without loss of generality, given a convolutional layer, let x, w, b be the input activations,

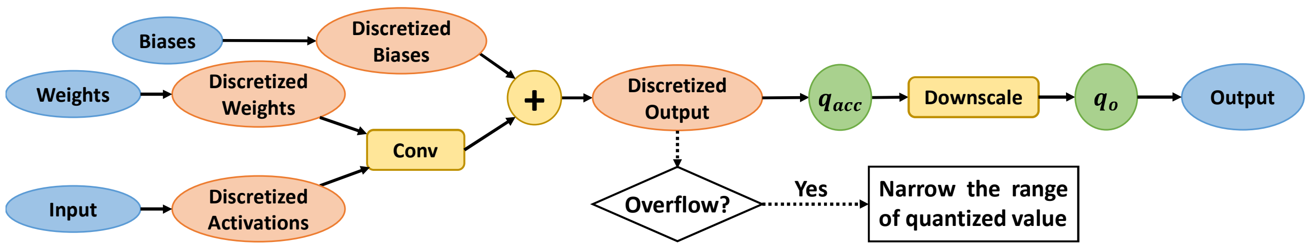

Fig. 1. An illustration of downscaling and overflow-aware training with simulated quantization of the convolution layer. First, we quantize the activations, weights, and biases to the discretized one and perform convolution to get the discretized output. Then, we convert the discretized output into high-precision integers q_{acc} (*e.g.*, 32-bit) by Eqn. 11 and downscale it to low-precision integers q_o (*e.g.*, 8-bit) to simulate the downscaling procedure. What's more, we narrow down the range of activations and weights' quantized value if overflow happens.

weight parameters, and bias parameters, respectively. Then, for k-bit quantization, we can apply the following function to quantize the activations and parameters to discretized ones:

$$\begin{aligned} D(x) &= s_x \cdot \text{clip}\left(Q(x, s_x), Q_{\text{low}}, Q_{\text{up}}\right), \\ D(w) &= s_w \cdot \text{clip}\left(Q(w, s_w), Q_{\text{low}}, Q_{\text{up}}\right), \\ D(b) &= s_x s_w \cdot \text{clip}\left(Q(b, s_x s_w), B_{\text{low}}, B_{\text{up}}\right), \end{aligned} \tag{2}$$

where function $\text{clip}\left(v, v_{\text{low}}, v_{\text{up}}\right)$ clips any number v into the range $[v_{\text{low}}, v_{\text{up}}]$. Here, Q_{up} and Q_{low} are the minimum and the maximum quantized values for activations and weight parameters, respectively, which are obtained by $Q_{\text{up}} = -Q_{\text{low}} = 2^{k-1} - 1$. B_{up} and B_{low} are the minimum and the maximum quantizad values for biased, respectively, which are obtained by $B_{\text{up}} = -B_{\text{low}} = 2^{B_a - 1} - 1$, and B_a is the bit-width for accumulator. We set the step size of weights s_w to $\max\left(\text{abs}\left(w\right)\right)/Q_{\text{up}}$ and learn the step size of activations s_x following [13]. Note that we quantize the x and w in a layer-wise and a channel-wise manner, respectively.

Quantized-aware Training. To improve the performance of the quantized model, we train the model with simulated quantization following [7]. The training approach can simulate the quantization effects during training. In this way, all weight parameters are stored in floating-point and can be updated in an end-to-end manner. For batch normalization layers, we folded the batch normalization parameters into the weights during the downscaling and overflow-aware training so that we could remove the batch normalization layers during inference.

Integer-arithmetic-only Inference. Following [7], all arithmetics in a convolution layer use integer arithmetic operations on the quantized values. We can get the corresponding quantized integer values of x, w, and b by

$$\begin{aligned} q_x &= Q(D(x), s_x), \\ q_w &= Q(D(w), s_w), \\ q_b &= Q(D(b), s_x s_w), \end{aligned} \tag{3}$$

where q_x and q_w are low precision activations and weights parameters(*e.g.*, 8-bit). Then, the intermediate results of matrix multiplication or convolution q_{acc} is accumulated in an accumulator as

$$q_{acc} = q_x * q_w + q_b, \tag{4}$$

where q_{acc} is high precision (*e.g.*, 32-bit). Following [7], the q_{acc} is downscaled from high precision intermediate results to low precision activations q_o by

$$q_o = M q_{acc}, \tag{5}$$

$$M = \frac{s_x s_w}{s_o}, \tag{6}$$

where q_o is the quantized integer input of the next layer, and s_o is the step size of the next layer's activation. In a layer with C output channels, the multiplication by floating-point value $M \in \mathbb{R}^C$ is implemented as a fixed-point multiplication as follows:

$$q_o = M q_{acc} \approx 2^{-n} M_0 q_{acc}, \tag{7}$$

where $M_0 \in \mathbb{R}^C$ are integer multipliers (*e.g.*, 32-bit) and multiplication by 2^{-n} is implemented with a bitshift operation.

B. *Downscaling-aware Training*

To make the models perform well with low bit-width multiplies, we propose a downscaling-aware training method to simulate the downscaling procedure of the convolution layer during training. When the bit-width of the multiplier is high enough (*e.g.*, 32-bit), the numerical error between q_o and $2^{-n} M_0 q_{acc}$ in the downscaling procedure (Eqn. 7) is so small that it can be ignored. Specifically, previous training methods ignore the downscaling procedure and convolute the discretized activations and parameters as follow:

$$D(o) = D(x) * D(w) + D(b), \tag{8}$$

where $D(o)$ is the floating-point convolution output of this layer and $*$ is the convolution operation.

However, when the bit-width of the multiplier is low (*e.g.*, 12-bit), the numerical error between q_o and $2^{-n} M_0 q_{acc}$ in Eqn. 7 will be relatively large, leading to significant performance degradation. Therefore, as shown in Fig. 1 we add a downscaling-aware procedure in the simulated quantization of the convolution layer. For a convolution layer, the multiplier M_0 and the n in Eqn. 7 is calculated as follows:

$$n = \min(\lfloor \log_2 \left(\frac{2^{B_m} - 1}{M} \right) \rfloor), \tag{9}$$

$$M_0 = \lfloor 2^n M \rfloor, \tag{10}$$

where B_m is the bit-width of the multiplier. During training, we simulate the downscaling procedure in Eqn. 7 as follows:

$$q_{acc} = \text{round}\left(\frac{D(x) * D(w) + D(b)}{s_x s_w}\right), \tag{11}$$

$$q_0 = \text{round}\left(2^{-n} M_0 q_{acc}\right). \tag{12}$$

Then, the floating-point convolution output $D(o)$ of this layer is given by:

$$D(o) = q_0 s_o. \tag{13}$$

In this way, during training, the quantized model can simulate the downscaling procedure. To avoid gradient vanishing issues, we use the Straight-Through Estimator (STE) [18] during the back-propagation of the rounding function because the rounding function is non-differentiable.

C. Overflow-aware Training

To avoid the overflow issue during inference, we use a floating-point factor $\alpha \geq 1$ to adjust the affine relationship between the real value range and the quantized value range following [11]. For example, in 8-bit quantization, we can quantize a real number r to a integer value range $[-127, 127]$ by default. With the factor α, r is rescaled to $[\lfloor -127/\alpha \rfloor, \lfloor 127/\alpha \rfloor]$. In this way, the quantized value range is narrowed down so that the numerical overflows are eliminated.

Specifically, we rescale the quantized value for each convolution or fully-connected layer. The step size s_x, s_w in quantization will be replace by $s_x{}'$, $s_w{}'$ as follows:

$$s_x{}' = \alpha s_x, \tag{14}$$

$$s_w{}' = \alpha s_w. \tag{15}$$

In fact, finding the proper factor α for each layer's activations and weights is difficult because we need to balance the performance and the numerical overflow. Intuitively, we increase the α of one layer if numerical overflow happens in this layer. Given the discretized output $D(o)$ of one layer, the batch size N_b, and the current factor α_t for this layer, we calculate the α_{t+1} as follows:

$$N_o = \text{overflow}\left(\text{round}\left(\frac{D(o)}{s_x{}' s_w{}'}\right), O_{min}, O_{max}\right), \tag{16}$$

$$\alpha_{t+1} = \alpha_t + \min\left(\eta \log\left(\frac{N_o}{N_b} + 1\right), \eta_{max}\right), \tag{17}$$

where $\text{overflow}(x, a, b)$ returns the number of element that is smaller than a or bigger than b, $O_{max} = -O_{min} = 2^{B_a - 1} - 1$ and B_a is the bit-width of the accumulator. η is the current dynamic learning rate of the model, η_{max} is a fixed learning rate. $O_{max} = -O_{min} = 2^{B_a-1} - 1$, where B_a is the bit-width of the accumulator. For efficiency, we calculate N_o and update the α every M steps (*e.g.*, 50 or 200 iterations).

D. Channel Pruning

To reduce the chip storage and take advantage of parallel computation, we use structure pruning methods to prune the channels of models before quantization. Specifically, we make sure that the preserved channels of each layer can be divisible by 32 so that the processor can handle a group of channels in parallel efficiently. Based on this guideline, we modify and use different channel pruning methods for different models on various tasks and datasets. We use DCP [16] to prune the MobileNetV2 and use CURL [17] to prune the ResNet-50 with their open source code. For Darknet-13 and VGG-16, we use Network Slimming [15] to pruned it. For ResNet18-RetinaNet and U-Net, we use the L2-norm of weight parameters as the importance metric to pruned it following [14]. Note that when the pruning rate is high (*e.g.*, 75%), DCP is not suitable to ResNet-50 because DCP only prunes the middle layer in the bottleneck structure of residual blocks, leading to an hourglass structure. In contrast, CURL [17] gets a shape similar to an opened wallet by pruning every layer in the residual blocks.

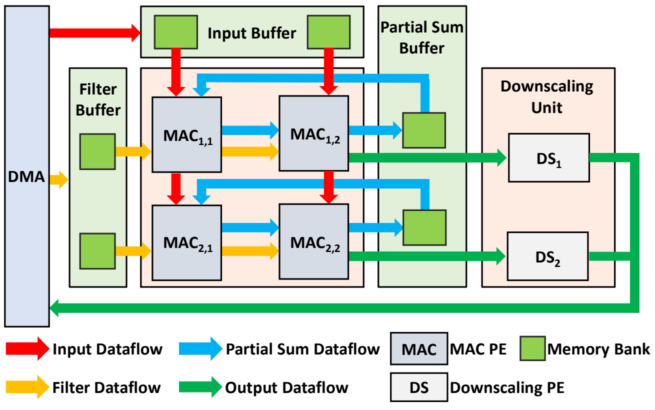

Fig. 2. The architecture of hardware accelerator. The accelerator is connected with Rocket chip SoC [19] with ROCC instruction interface and Tilelink data interface. The accelerator is composed of the MAC array for convolution operation, the downscaling units, and on-chip memory banks. The MAC array is implemented with a weight-stationary systolic array.

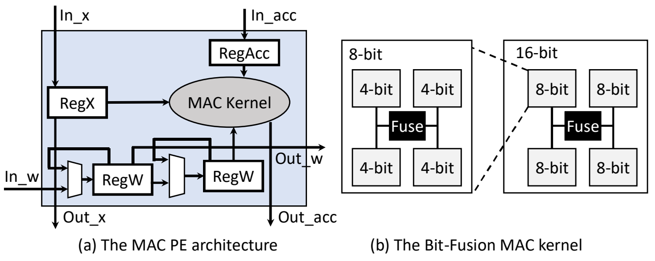

Fig. 3. The architecture of MAC PE and MAC kernel.

IV. Agilely Customizable Processors and SoCs

Fig. 2 shows the architecture of the hardware accelerator for our compressed DNN models, while Fig. 3 (a) shows the architecture of the MAC PE. The weight data is sent to each MAC PE with systolic dataflow and updated periodically, and input data flows across the MAC array with vertical systolic dataflow. The partial sum is first accumulated in the channel dimension with reduce-and-forward dataflow from left to right, and then stored to the partial sum buffer before being sent back to the MAC array to accumulate in the kernel dimensions.

TABLE II
THE TOP-1 ACCURACY OF COMPRESSED MODELS ON IMAGE CLASSIFICATION.

Model	Dataset	#FLOPs ↓ (%)	Acc. (%)		
			FP32	INT16	INT8
VGG-16 [1]	CIFAR-100	0	74.5	74.5	73.9
		50.6	72.3	71.7	71.8
ResNet-50 [2]	ImageNet	0	76.0	75.9	75.7
		72.5	72.7	73.1	72.6
		81.0	71.7	72.1	71.8
		90.1	65.1	65.7	65.2
MobileNetV2 [3]	ImageNet	0	71.9	72.7	72.4
		33.8	71.0	70.8	70.8
		45.7	69.2	69.2	68.7
		53.4	67.2	65.7	65.5
		72.4	59.3	59.2	55.9

TABLE III
THE MAP (IOU=0.50) OF COMPRESSED MODELS ON OBJECTION DETECTION.

Model	Dataset	#FLOPs ↓ (%)	mAP@0.5 (%)		
			FP32	INT16	INT8
Darknet-13 [4]	COCO	0	33.8	34.7	33.5
		27.0	32.3	33.9	32.2
ResNet18-RetinaNet [5]	PASCAL	0	53.1	53.4	52.7
		38.7	52.4	51.9	52.0
		40.9	51.9	51.8	51.8
		60.4	49.1	51.5	51.1

TABLE IV
THE MIOU OF COMPRESSED MODELS ON SEMANTIC SEGMENTATION.

Model	Dataset	#FLOPs ↓ (%)	mIoU		
			FP32	INT16	INT8
U-Net [6]	Cityscapes	0	0.591	0.601	0.595
		76.3	0.573	0.582	0.579
		88.6	0.550	0.560	0.558

TABLE V
EFFECT OF OVERFLOW-AWARE AND DOWNSCALING-AWARE TRAINING ON CITYSCAPES. "OWA" DENOTES OVERFLOW-AWARE TRAINING, "DSA" DENOTES DOWNSCALING-AWARE TRAINING. "NO OVERFLOW" INDICATES NO NUMERICAL OVERFLOW HAPPENS IN THE TEST SET.

Model	OWA	DSA	No Overflow	mPA (%)	mIoU
U-Net [6]	×	×	×	60.61	0.212
	✓	×	✓	64.41	0.293
	×	✓	×	88.37	0.547
	✓	✓	✓	**88.53**	**0.558**

Finally, when the accumulation finishes, the output data enters the downscaling unit to reduce the bit-width, and is stored back to memory with DMA.

Both the MAC unit and the downscaling unit can be configured with flexible bit-width. Fig. 3 (b) shows the MAC kernel implemented with Bit-Fusion technology [20]. The MAC unit is controlled by a configure register to select the desired output with different precision requirements from 4-bit to 16-bit. The MAC units in the downscaling PE are implemented similarly.

V. EXPERIMENTS

We conducted two sets of experiments, one shows the effectiveness of our method on common vision tasks (Sec. V-A), and the other evaluates the effectiveness of different components in our method (Sec. V-B). Here, for 8-bit quantization, we set the bit-widths of accumulator B_a and multiplier B_m to 16-bit and 12-bit, respectively. For 16-bit quantization, we set the bit-widths for accumulator B_a and multiplier B_m to 32-bit and 12-bit, respectively. "FP32" denotes the full-precision network, while "INT16", "INT8" represent quantizing the activations and weight parameters to 16-bit and 8-bit integers, respectively. "#FLOPs ↓ (%)" denotes the percentage of floating-point operations that are pruned by channel pruning.

A. Results on Common Vision Tasks

Results on Image Classification. We apply our method to compress VGG-16 [1], ResNet-50 [2] and MobileNetV2 [3] and evaluate the performance on CIFAR-100 and ImageNet dataset. We report the Top-1 accuracy of full-precision models and quantized models under different pruning rates in Tab. II. From the results, compared with the full-precision model, 8-bit quantized models achieve comparable performance. For example, with FLOPs reduction of 90.1%, our compressed ResNet-50 with 16-bit and 8-bit quantization even outperforms the full-precision one. For MobileNetV2 with FLOPs reduction of 33.8%, our obtained 8-bit MobileNetV2 only results in 0.2% performance drop.

Results on Objection Detection. To evaluate our method on objection detection, We apply the proposed method to compress Darknet-13 [4] and ResNet18-RetinaNet [5] on COCO2017 [21] and PASCAL datasets and report the FLOPs reduction and mAP (IoU=0.50) in Tab. III. From the results, the 8-bit quantized models achieve comparable performance with the full-precision one. With FLOPs reduction of 27%, the 16-bit compressed Darknet-13 outperforms the full-precision one by 1.6%. Besides, the mAP of the 16-bit and 8-bit compressed ResNet18-RetinaNet are higher than the full-precision one with FLOPs reduction of 60.4%. It further demonstrates the effectiveness of our method.

Results on Semantic Segmentation. We apply our method to compress U-Net [6] and report the FLOPs reduction and mean intersection over union (mIoU) on the Cityscapes dataset in Tab. IV. From the results, our method achieves a better performance even for a compact model. For example, with FLOPs reduction of 88.6%, our method still obtains better performance than the full-precision model.

B. Ablation Studies

Effect of Overflow-aware and Downscaling-aware. To investigate the effect of the overflow-aware and downscaling-aware training, we apply different combinations of training strategies on compressed 8-bit U-Net and report the mean pixel accuracy (mPA) and mIoU on Cityscapes in Tab. V. Here, the baseline that does not use overflow-aware and downscaling-aware training has a poor performance and suffers from the overflow issue. Both training strategies can improve the performance, and we obtain the compressed model with the best performance when we use two training strategies jointly.

TABLE VI
RESULTS OF U-NET WITH DIFFERENT BIT-WIDTHS FOR THE ACCUMULATOR ON CITYSCAPES.

B_a	10	12	14	16	18	20
mIoU	0.006	0.546	0.554	0.558	0.555	0.555

TABLE VII
RESULTS OF U-NET WITH DIFFERENT BIT-WIDTHS FOR THE MULTIPLIER ON CITYSCAPES.

B_m	2	3	4	8	12	18	32
Area (μm^2)	543	687	865	1519	2109	2746	4343
Power (μW)	14.20	16.29	17.26	24.76	31.20	39.41	57.63
mIoU	0.036	0.530	0.554	0.558	0.558	0.559	0.558

Effect of Different Bit-widths for the Accumulator. To evaluate the effect of B_a, we report the mIoU of compressed 8-bit U-Net on Cityscapes under different B_a in Tab. VI while using 12-bit multipliers. When the bit-width for the accumulator is bigger than 14, the performance is stable, meaning that a 14-bit accumulator is sufficient for this compressed model. However, when the bit-width for the accumulator is smaller than 12, the performance drops dramatically because the quantized value range for activations and parameters is too small, which limits the representation power of the quantized models.

Effect of Different Bit-widths for the Multiplier. To investigate the effect of B_m, we compare the compressed 8-bit UNet on Cityscapes under different B_m while using 16-bit accumulators. We report the chip area and the dynamic power for downscaling procedure, and the mIoU of compressed models in Tab. VII. When the bit-width for the multiplier becomes lower, the chip area and the dynamic power for downscaling process become smaller. However, when the bitwidth of the multipliers is smaller than 3, the model can not converge.

C. Characteristics of the Agilely Customizable Processor

In this part, we conduct relevant simulation experiments to investigate the characteristics of the designed agilely customizable processor in a commercial 55 nm CMOS technology. The processor infers 8-bit quantized models with 16-bit accumulators and 12-bit multipliers, and infers 16-bit quantized models with 32-bit accumulators and 12-bit multipliers. According to the simulation results, the energy consumption measured at the voltage of 1V and 25 degrees Celsius is about 2.57 W. The energy consumption measured at the voltage of 0.8 V and 25 degrees Celsius is about 1.54 W. The maximum achievable throughput of the 8-bit fixed-point operation is up to 10.4858 TOPS. At the voltage of 0.8 V and 25 degrees Celsius, the energy efficiency ratio can reach up to 5.68 TOPS/W.

VI. CONCLUSION

In this paper, we have proposed a downscaling and overflow-aware model compression scheme for efficient vision processors. Specifically, the proposed downscaling-aware training simulates the downscaling procedure, and the overflow-aware training adaptively determines the quantized value range for activations and parameters. In this way, the models are adjusted to perform with low bit-width multipliers, and prohibit numerical overflow with low bit-width accumulators as much as possible. We also conducted channel pruning to restrict the channel number of models for better parallel computing. Based on this scheme, we have further co-designed a vision processor and its SoC to demonstrate the proposed scheme. Experiments on various vision tasks have shown that our method is able to achieve significant chip area reduction on vision processors with low bit-width accumulators and multipliers while preserving performance.

ACKNOWLEDGMENT

This work was partially supported by the Key-Area Research and Development Program of Guangdong Province (2019B010155002, 2018B010107001), National Natural Science Foundation of China (NSFC) 62072190.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.
[4] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, pp. 2980–2988, 2017.
[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, pp. 234–241, Springer, 2015.
[7] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.
[8] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *ISSCC*, pp. 10–14, IEEE, 2014.
[9] S. Han, X. Liu, H. Mao, J. Pu, *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
[10] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *VLSI*, vol. 27, no. 8, pp. 1874–1885, 2019.
[11] H. Xie, Y. Song, L. Cai, and M. Li, "Overflow aware quantization: Accelerating neural network inference by low-bit multiply-accumulate operations," in *IJCAI*, pp. 868–875, 2021.
[12] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Towards effective low-bitwidth convolutional neural networks," in *CVPR*, 2018.
[13] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *ICLR*, 2020.
[14] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2018.
[15] Z. Liu, J. Li, Z. Shen, G. Huang, *et al.*, "Learning efficient convolutional networks through network slimming," in *ICCV*, pp. 2736–2744, 2017.
[16] J. Liu, B. Zhuang, Z. Zhuang, Y. Guo, *et al.*, "Discrimination-aware network pruning for deep model compression," *TPAMI*, 2021.
[17] J.-H. Luo and J. Wu, "Neural network pruning with residual-connections and limited-data," in *CVPR*, pp. 1458–1467, 2020.
[18] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.
[19] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, 2016.
[20] H. Sharma, J. Park, N. Suda, L. Lai, *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *ISCA*, pp. 764–775, IEEE, 2018.
[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, *et al.*, "Microsoft coco: Common objects in context," in *ECCV*, pp. 740–755, Springer, 2014.