A 1.15-TOPS 6.57-TOPS/W Neural Network Processor for Multi-Scale Object Detection with Reduced Convolutional Operations

Reiya Kawamoto, Masakazu Taichi, Masaya Kabuto, Daisuke Watanabe, Shintaro Izumi, *Member, IEEE*, Masahiko Yoshimoto, *Member, IEEE*, Hiroshi Kawaguchi, *Member,* IEEE, Go Matsukawa, Toshio Goto, and Motoshi Kojima

Abstract—**For automated driving cars, we present a 40-nm dedicated object detection processor with only three operations: 3 × 3 convolution, 1 × 1 convolution, and 4 × 4 deconvolution. Multi-scale object detection at high recognition accuracy is possible by virtue of the deconvolution feature and concatenation. The input memory for a feature map has 8-bit width. A multiplier for the inputs has 8-bit precision. Partial-sum memory, however, has 16-bit width to suppress detection accuracy deterioration in a layer with 1024 channels in the target network. By fixed-point bit precision, the external memory bandwidth and internal memory capacity are reduced. Optimized parallelization in input and output channels reduces the external memory bandwidth to 0.062 billion accesses per 1280 × 384 image with internal memory capacity of 400 kB. The detection error is 1.9% of that using single-precision floating point. The maximum operating frequency is 500 MHz at a supply voltage of 1 V. Its peak performance is 1.15 TOPS. The maximum energy efficiency is 6.57 TOPS/W at 174 MHz and 0.6 V.**

Index Terms—**Automated driving, Convolutional neural network, Deconvolution, Multi-scale object detection**

I. Introduction

In recent years, convolutional neural networks (CNNs) have become the most powerful and widely used method for computer vision. Various algorithms using CNNs have been developed for object detection [1−9]. Their performance is improving year by year. However, state-of-the-art CNN-based object detection models are becoming larger and deeper than earlier ones because the number of parameters becomes enormous, thereby requiring many memory resources. Storing all weights and feature maps on on-chip memory is becoming impossible. A CNN requires vast amounts of data access. Therefore, communication with external memory (usually DRAM) becomes indispensable and comes to constitute a bottleneck. Memory bandwidth is a crucially important issue for embedded systems. Moreover, access to off-chip DRAM consumes a hundred times the energy, or more, than on-chip SRAM [10]. Reusing data and thereby reducing memory access with off-chip DRAM to the greatest extent possible is important. To reduce the memory bandwidth, earlier works have proposed CNN dataflows [11]. Typically, access to off-chip DRAM can be suppressed by keeping weights (weight stationary) and partial sums (row stationary / output stationary) and by reusing them to the greatest extent possible in local memory.

Enormous computation costs are another issue that one must confront with CNN. For inference, using a fixed point instead of a floating point is a common approach used with embedded systems. Reducing bit precision saves chip power and area both for logic circuitry and memory. Low-precision operations also reduce energy and area costs: an 8-bit fixed point (INT8) adder consumes 1/30 of the energy and occupies only 1/116 of the area of a 32-bit floating point adder. Also, an INT8 multiplier consumes 1/18.5 of the energy and 1/27.5 of the area [12]. State-of-the-art CNN accelerators have reduced input and weight precision to 1−16 bits [13−28].

As described herein, we propose a multi-scale object detection processor that operates at low power for the automated driving of cars. The multi-scale object detection techniques are extremely important steps toward the realization of automated driving of cars. Hundreds of in-vehicle electronic control units (ECUs) are installed in automobiles. Their high performance and high efficiency are required. The remainder of this paper is organized as follows. Section II describes related works. Section III presents the target network using only three convolutional operations (reduced convolutional operations: 3 × 3 convolution, 1 × 1 convolution, and 4 × 4 deconvolution for simple hardware). Some techniques to reduce memory bandwidth are described in Section IV. Sections V and VI present the proposed 8-bit processor design and its measurement results. The final section presents the relevant findings.

This study was supported by the grant from Toyota Motor Corporation.

Reiya Kawamoto, Masaya Kabuto, Daisuke Watanabe, and Hiroshi Kawaguchi are with The Graduate School of Science, Technology and Innovation, Kobe University, 1-1 Rokkodai-Cho, Nada-Ku, Kobe, 657-8501, Japan (e-mail: kawamoto.reiya@cs28.cs.kobe-u.ac.jp).

Masakazu Taichi, Shintaro Izumi, and Masahiko Yoshimoto are with The Graduate School of System Informatics, Kobe University, 1-1 Rokkodai-Cho, Nada-Ku, Kobe, 657-8501, Japan.

Go Matsukawa, Toshio Goto, and Motoshi Kojima are with Toyota Motor Corporation, 1 Toyota-Cho, Toyota, Aichi, 471-8571, Japan.

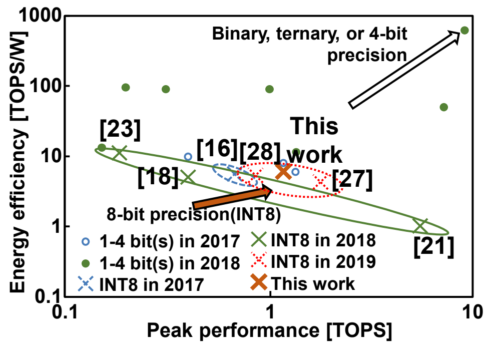


Fig. 1 Energy efficiency versus peak performance in CNN processors presented in the ISSCCs and Symposia on VLSI Circuits in 2017–2019.

II. Related Work

A. Dataflows for CNN

Memory access is a bottleneck in processing CNN. Various dataflows have been proposed for less memory bandwidth: Vinayak et al. proposed "nn-X" implemented with weight stationary dataflow [29]. The weight stationary dataflow stores weights from DRAM in a register file (RF), and reads from the RF to the greatest extent possible to minimize memory bandwidth reading burdens. Du et al. proposed "ShiDianNao," designed with output stationary dataflow [30], which keeps partial sums in the RF, thereby minimizing bandwidth in reading and writing the partial sums. Other than optimizing only weights or only partial sums, row stationary dataflow was proposed to reuse all weights, partial sums, and activations [31]. We basically adopt weight stationary dataflow for the proposed processor, yet it is optimized for combination of input and channel processing in our architecture. Its dataflow strategy is discussed in Subsection IV.B.

B. Conventional accelerators for CNN

Reduction of bit precision is also important for reducing memory bandwidth and limiting computation costs. Nevertheless, a tradeoff exists between bit length and accuracy. Earlier methods used a fixed point with static quantization. This strategy uniquely sets a fractional length and an integer length in fixed-point operations. However, in CNN, dynamic ranges of weights, partial sums, and activations differ among layers. In recent years, a dynamic fixed-point strategy has been adopted by analyzing a dynamic range in each layer [13]. We take the dynamic fixed-point strategy as well as other CNN processors (Subsection IV.A for details). Fig. 1 shows that recently presented exquisite CNN processors with low bit precision have improved energy efficiency and peak performance [13–28]. Nevertheless, many use binary (ternary or 4-bit) precision, which is not amenable to multi-scale object detection. For convolutional operations, our CNN processor uses INT8 instead for high accuracy.

C. CNN for object detection

Various approaches for object detection have been proposed in recent years. Their operating speeds and accuracies have improved rapidly. Region-CNN (R-CNN) [1], a pioneering method, applies a CNN to object detection. Numerous region proposals where objects might exist are extracted from an image by Selective Search [2]. Then, after the region proposals are reshaped to a fixed size, object detection is performed by calculating the feature quantity for each region proposal. Nevertheless, execution time is very slow because R-CNN must carry out three processes: the CNN, support vector machine (SVM) classification, and bounding-box regression. Spatial pyramid pooling (SPP) [3] has shortened the processing time for object detection with pyramid feature maps: only a single convolution process is done for an entire image. Fast R-CNN [4] applies back propagation to all layers by introducing multi-task loss. It takes simple region-of-interest (ROI) pooling instead of SPP to resize a feature map. Faster R-CNN [5] proposes a region proposal network (RPN) instead of Selective Search to extract region proposals. This method enables end-to-end training. By introducing a concept of anchors, objects with different aspect ratios are classifiable after detecting their region proposals. "You only look once" (YOLO) [6] realizes localization and classification simultaneously with a single CNN. After the entire image is divided into grids, classification and a bounding box of an object are obtained on the grid bases. It can operate at high speed because of its simple CNN structure, although it is inferior in terms of accuracy, especially for an object smaller than a grid. The entire image is used during training. Therefore, false detection of the background is suppressed. Single shot multibox detector (SSD) [7] is aimed at a different approach using a single network. SSD estimates object recognition and bounding box offset by convolution layers with small filter sizes. Multi-scale detection can be accomplished by detecting objects from multiple aspect ratios in feature maps with different scales. Unfortunately, the recognition rate of a small object is as low as SSD because a feature map of a shallow layer has less semantic information. To resolve this shortcoming, deconvolutional SSD (DSSD) [8] added deconvolution layers to draw a high-level context for detection. Feature fusion SSD (FSSD) [9], a recent derivative of SSD, further introduced fusion of multiple convolutional/deconvolutional layers of SSD: concatenation, element-sum, and element-product are proposed as a feature fusion module. Our target network is inspired by FSSD with concatenation.

TABLE I KITTI detection results in different models (FP32 simulation).

	Target network	Max pooling	Average pooling
Accuracy	88.3	89.5	89.9

III. Target Network

Fig. 2 presents the target network model. Its 63 layers include concatenation layers. Each convolution layer consists of either a 3 × 3 or a 1 × 1 kernel. A pooling layer is often used to reduce the feature map size, although implementing the pooling layer requires extra hardware (typically adders) and increases the memory access. In our target network, the feature map size is reduced by a factor of four with a stride of two. Moreover, its accuracy loss is less than 1.6% compared to max pooling and

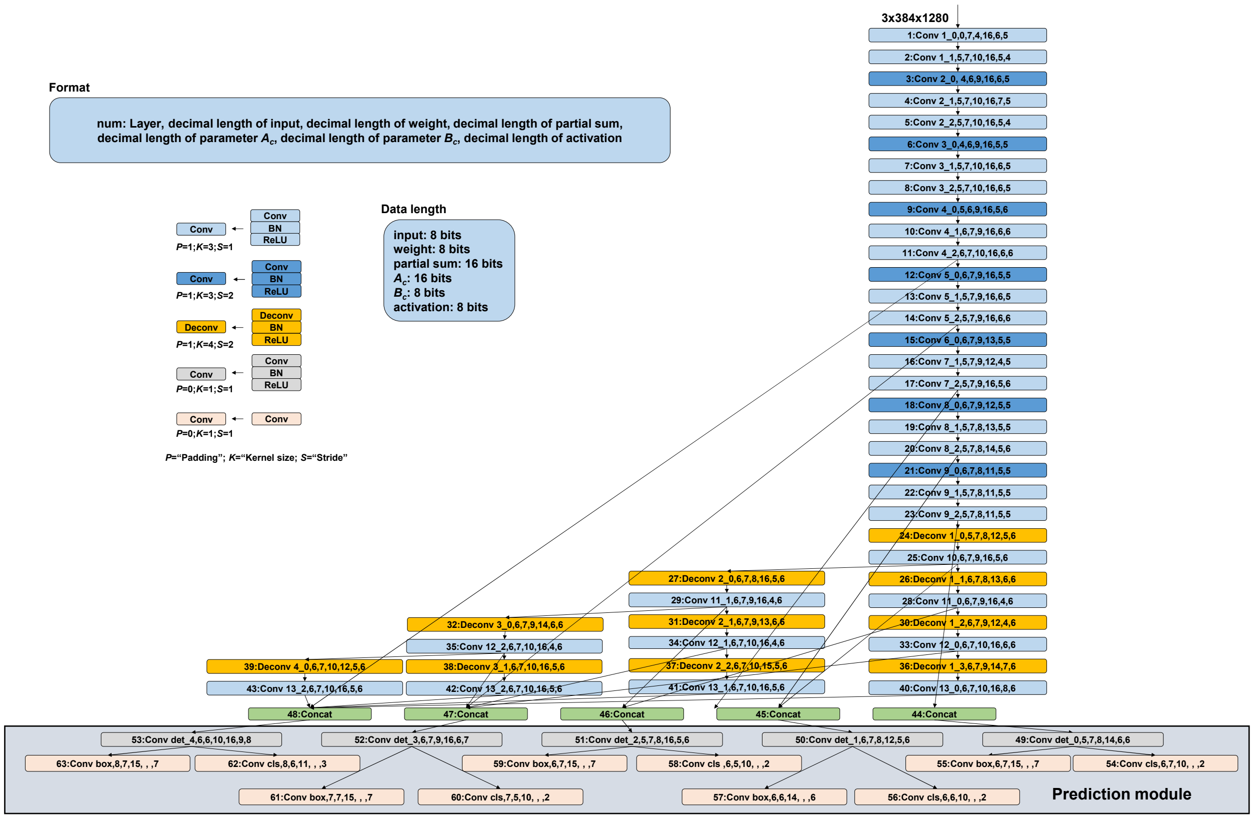


Fig. 2 Target network model. *P* means padding. *K* and *S* respectively signify the kernel size and stride.

average pooling (TABLE I). Deconvolutional layers for multi-scale object detection have a 4 × 4 kernel. Batch normalization [32] is performed in the convolution and deconvolution layers (BN in the figure), except for the final output layers. The activation function is a hardware-friendly rectified linear unit (ReLU) [33]. The module for combining context information refers to the concatenation module proposed in FSSD. However, unlike FSSD, we add batch normalization to the deconvolution layers. The prediction module outputs candidate bounding boxes and classes.

An input image has a KITTI [34] size of 1280 × 384 pixels with three channels: it is much larger than ImageNet [35]. KITTI is a dataset used for automobile vision. The computational amount of the entire network is 26.9 GOPS for a KITTI-size input image. The network detects cars, trucks, buses, motorcycles, bicycles, people, riders, and other objects.

IV. Memory Bandwidth Reduction

On a CNN processor, the computation amount is constant once a network is fixed. However, the memory bandwidth must be considered carefully because memory accesses for kernel weights, batch normalization parameters (shifts and scales), feature maps (inputs), partial sums (interim results), and activations (outputs) depend strongly on the computational procedure in channel directions in a layer. Furthermore, bit precision for the model strongly affects the memory bandwidth.

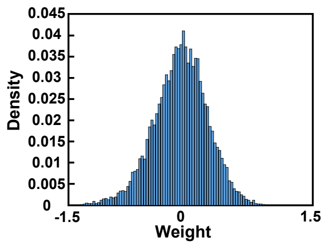


Fig. 3 A histogram of weights in a layer.

A. Fixed-point bit precision

For the original model implemented by Caffe [36], all operations were conducted with single-precision floating points (FP32). In our design, feature maps, activations, weights, and shifts are converted to 8-bit precision integers (INT8) from the original Caffe model. The other ones have 16-bit integers: scales and partial sums. For inference with a small network such as The Visual Geometry Group's 16-layer model (VGG16) [37], accuracy sometimes becomes even better at the entire INT8 than that for FP32 [38]. However, because our target network comprises more than 60 layers, the accuracy degradation would accumulate greatly if it was calculated only with INT8; we carefully adjust their bit precisions for the parameters. All the parameters have different dynamic ranges for each layer. Therefore, we take the dynamic fixed-point strategy. Their dynamic ranges in each layer are analyzed. Also, their bit lengths in a fractional part are set so that a sum of absolute differences between FP32 and INT8 is minimized [39].

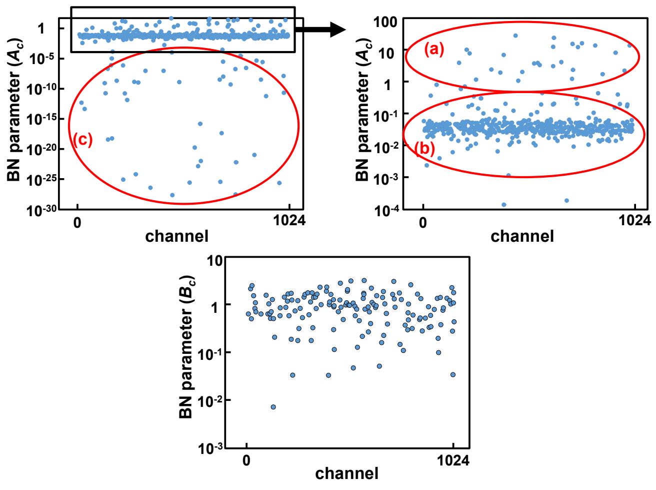


Fig. 4 Distributions of batch normalization parameters A_c and B_c on KITTI dataset.

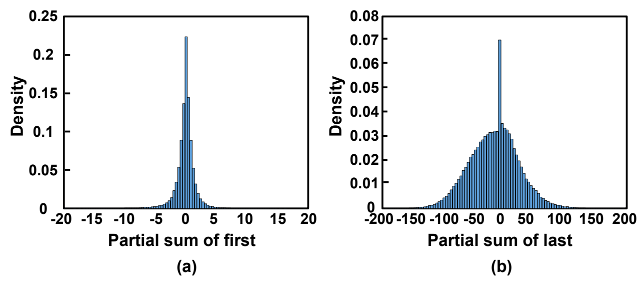


Fig. 6 Transition of the distribution of partial sums.

1) *Kernel weights*

A histogram of weights in a shallow layer is depicted in Fig. 3. In the figure, the precision is FP32. It exists mainly in the range of −1.5 to 1. As a layer deepens, the weights converge to zero. Moreover, the proportion that is near zero increases as the layer gets deeper. Therefore, some deeper layers require no integer part. They only have a decimal part. We carefully select the digit numbers in the integer and decimal parts in every layer to maintain accuracy, based on [39] (decimal lengths are shown in Fig. 2). Then the weights are tuned by retraining with Ristretto [40].

2) *Batch normalization parameters*

In a CNN platform such as Caffe, normalization is conducted after adding a bias to the convolution result. Then, a shift and scale are calculated. This successive procedure requires multiple multiply and-accumulate (MAC) operations. The biases and parameters for batch normalization are determined at the training time. Consequently, they are all constant at an inferencing time; once the constants are fixed, the batch normalization can be conducted as a single MAC operation, as expressed in the following (1), where $x_{i,c}$ is a convolution result and $y_{i,c}$ is an activation. Also, subscripts i and c respectively denote a coordinate and a channel. The multiplication and accumulation coefficients A_c and B_c can be prepared as constants in (2) and (3). Therein, b_c represents a bias. The batch normalization parameters are given as average μ_c, standard deviation σ_c, shift β_c, scale γ_c, and correction term ε.

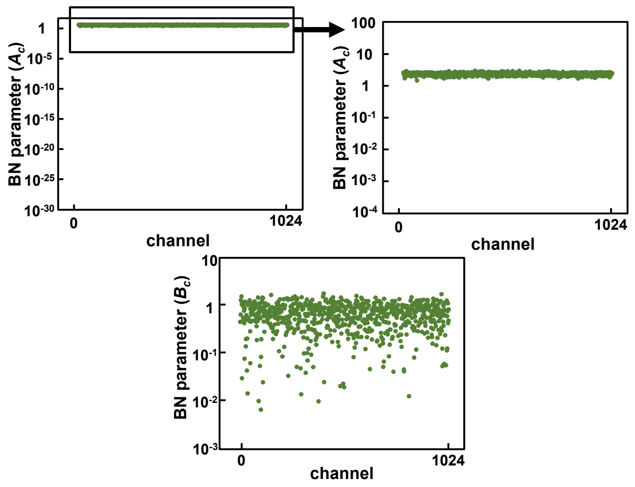


Fig. 5 Distributions of batch normalization parameters A_c and B_c on the BDD dataset.

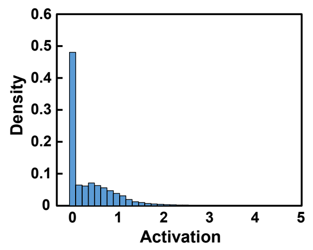


Fig.7 Histgram of activation in a layer.

$$y_{i,c} = A_c \cdot x_{i,c} + B_c \tag{1}$$

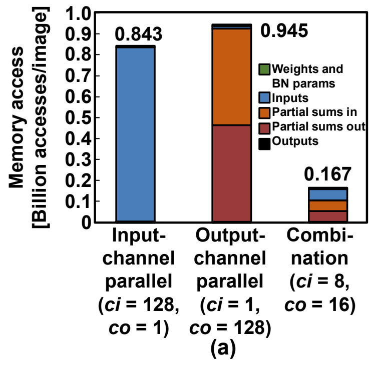
$$A_c = \frac{\gamma_c}{\sqrt{\sigma_c^2 + \varepsilon}} \tag{2}$$

$$B_c = \frac{\gamma_c}{\sqrt{\sigma_c^2 + \varepsilon}}(b_c - \mu_c) + \beta_c \tag{3}$$

An example of distributions for A_c and B_c in a layer at FP32 is portrayed in Fig. 4: the KITTI dataset. Particularly, A_c is broadly scattered and categorized into three parts: (a) minority but strengthening the convolution result ($1 < A_c < 100$); (b) weakening the convolution result ($0.01 < A_c < 1$) but majority; and (c) negligible ($A_c < 0.01$). Actually, (a) and (b) are distributed over a wide dynamic range of 10^4 (= 0.01–100), although both are important. We set the bit width of A_c to 16-bit to cover the wide dynamic range and to suppress accuracy degradation. In contrast, B_c is concentrated around one; it is not distributed as widely as A_c. Furthermore, B_c is not so dominant a parameter as A_c. We keep B_c at INT8. Actually, A_c and B_c depend on a dataset. Fig. 5 portrays the distribution for A_c and B_c trained with the Berkeley DeepDrive (BDD) dataset, which shows different aspects from Fig. 4 even in the same layer but with less dynamic range. Therefore, it can be considered that 16-bit A_c and 8-bit B_c are sufficient.

3) *Partial sums*

The target network model has a layer with 3–1024 channels. Fig. 6 presents a transition of partial sum's distributions in a layer with 512 input channels. In the example, 512 input channels are divided into 64 times in partial-sum calculation: The partial sum is accumulated in every eight input channels. Figs. 6(a) and 6(b) respectively portray distributions of the first



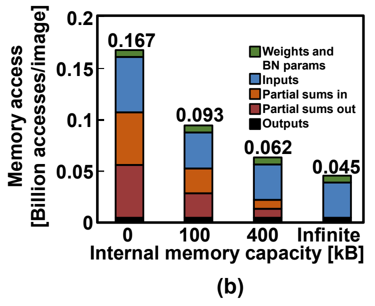


Fig. 8 (a) Memory access tradeoff in channel directions and (b) memory bandwidth when the internal memory capacity is varied. An access to DRAM has a 64-bit width.

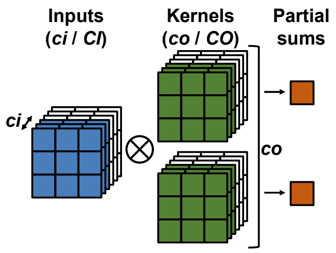


Fig. 9 Channel division: *CI* and *CO* respectively signify the total numbers of input channels and output channels.

and last (= 64th) partial sums. The partial sums swell to a large value particularly when the input has 512 channels or more. We set the bit length of the partial sum to 16 bits for the large number of input channels.

4) *Activations*

The distribution of activations in a layer in the target network model is depicted in Fig. 7. It is suppressed mainly in a small range (0–2) by virtue of batch normalization and ReLU. Therefore, the bit length of activations is set back to INT8 as a final output of the layer, although the partial sums have 16-bit precision. Unlike other parameters, zero or signed values can be taken by adapting the ReLU treatment; it is not necessary to consider a sign bit in an activation.

B. Memory access

Fig. 8(a) portrays the tradeoffs among input-channel parallelization, output-channel parallelization, and their combination. When parallel processing is executed along with the input channels, a single partial sum is calculated using one computation. A bandwidth for the partial sums is reduced, but inputs must be reread frequently from external memory. In contrast, partial sums are repeatedly output and input in the

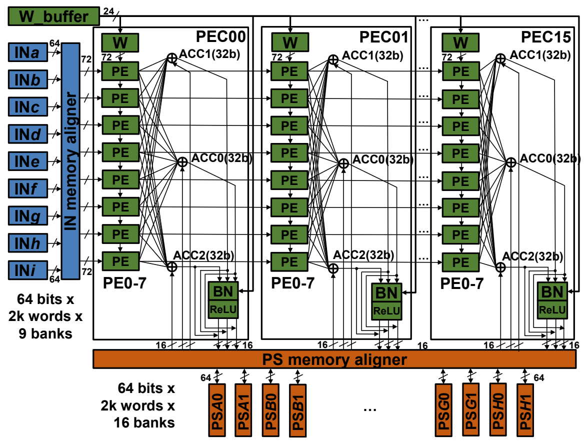


Fig. 10 Processor architecture.

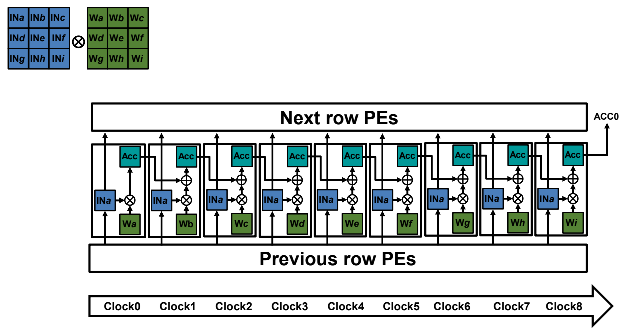


Fig. 11 Block diagram of a 3 × 3 systolic array as a reference.

output-channel parallelization. Our design takes a combination of them. In doing so, it is possible to suppress both memory accesses of the inputs and the partial sums. Then they are not lopsided to either side. The memory access is minimized to 0.167 billion accesses per image when an input-channel parallelization degree *ci* = 8 and an output-channel parallelization *co* = 16 in Fig. 9. The external memory access can be reduced further using internal memory as a buffer; if sufficient capacity were secured as portrayed in Fig. 8(b), it would come to 0.045 billion accesses per image. Our design takes an internal memory capacity of 400 kB (16 kB × 25 banks) and external memory accesses of 0.062 billion per image.

V. Processor Design

Fig. 10 portrays a block diagram of the processor architecture. The input (IN) memory and partial-sum (PS) memories are, respectively, 144 kB (16 kB × 9 banks) and 256 kB (16 kB × 16 banks), which is 400 kB in all, as discussed above. Every memory has 64-bit width per word (8 × 8 bits). Kernel weights (W in the figure) and batch normalization parameters (BN) are implemented with FIFOs (W_buffer) in each processor element cluster (PEC) to perform convolution, deconvolution, and batch normalization. A 3 × 3 convolution occupies more than 50% of the network operations. 64-bit inputs (an 8-bit feature map × 8 channels) are read out from IN memories. The IN memory aligner feeds eight 3 × 3 inputs through 16 PECs, which means that *ci* = 8 and *co* = 16 in this architecture. Eight processor elements (PEs) in a PEC correspond to parallel processing in the input channel direction. Also, 16 PECs correspond to parallel processing in the output direction. In shallow layers

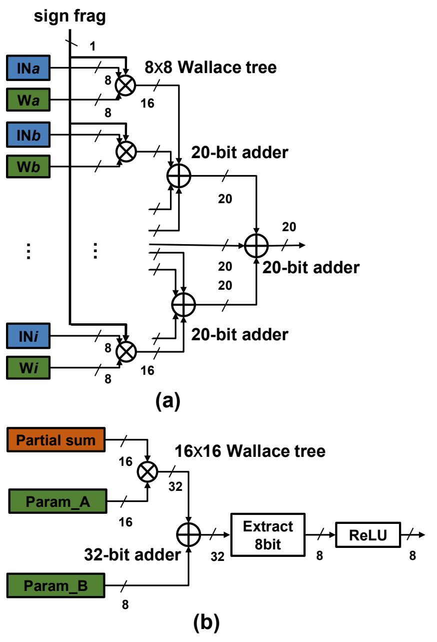


Fig. 12 (a) 8-bit convolution MACs and (b) a 16-bit batch normalization MAC.

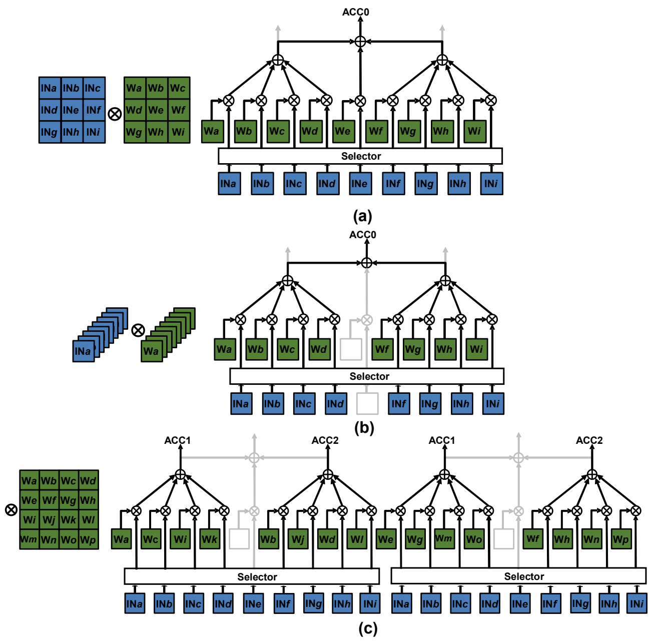


Fig. 13 PE configurations for (a) 3 × 3 convolution, (b) 1 × 1 convolution, and (c) 4×4 deconvolution.

near an image input, an input channel is less than 8, in which case some PEs do not work. Similarly, when an output channel is less than 16, some PECs do not work. For example, half of the PEs and half of the PECs do not work in a layer where *ci* = 4 and *co* = 8; its core utilization results in 25%.

A. PEs: processor elements

Fig. 11 shows a block diagram of a commonly used systolic array with 9-grid MAC units [41]. In the conventional systolic array for a 3 × 3 convolution operation, input IN*a* and weight W*a* are multiplied at the first clock cycle; then, in the next clock cycle, IN*b* × W*b* are accumulated at the next clock cycle. The data flow is repeated to the right end. The systolic array covers various MAC operations. It is therefore versatile, but with high latency. Each grid is associated with a MAC operation and is on simple data paths. However, it has several pipeline registers and accumulators for the data path, necessitating complicated control. In our design presented in Fig. 13(a), a PE is implemented straightforwardly as SIMD with an adder tree with a three-stage pipeline; it is not versatile but it is dedicated for 3×3 convolution operations. As shown in Figs. 12(a) and 12(b), for processor design simplicity, adders of only two types (four-input 20-bit and two-input 32-bit adders) are used. As a multiplier, an 8 × 8 Wallace tree is used for 8-bit convolution; four 8 × 8 multipliers are used for 16-bit batch normalization in a PEC. Only the input to the first layer is signed: the others are all unsigned as inputs. Therefore, the eight-bit multiplier supports both a signed input × a signed weight and an unsigned input × a signed weight. Input data and weights with 8-bit fixed-point precision are multiplied. They generate 16-bit output. Their outputs are accumulated with 20-bit precision, which is forwarded to the PE output (Fig. 12(a)). Then, in a PEC, eight PE outputs are accumulated as a partial sum with 32-bit precision (Fig. 10). Finally, the 32-bit partial sum is reduced to 16-bit precision and is stored in PS memory. For a final activation, 32-bit precision output is generated by the batch normalization; then it is reduced to 8-bit precision (Fig. 12(b)).

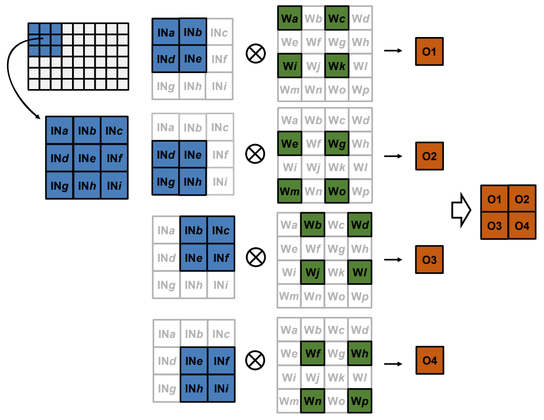


Fig. 14 4 × 4 deconvolution in the processor design.

In a PE, a kernel is shifted corresponding to a stride downward from the upper left end in the input feature map. When the bottom end is completed, the kernel is applied sequentially again from the top of the next column. At this time, newly required inputs must be read again from the IN memory. In this way, inputs are aligned with the IN memory aligner.

B. IN memory aligner

The IN memory aligner receives 576 bits (8 bits × 8 channels × 9 banks) from IN memories. It then forwards them to each input channel (8 bits × 9 pixels × 8 PEs) through 8 PEs in 16 PECs. The input channels are common in PEC0 – PEC15. Fig. 13(a) portrays a PE configuration for the 3×3 convolution: nine MACs are executed for a single-input channel. Although the PE is optimized for 3 × 3 convolution, it can be exploited for 1 × 1 convolution. Fig. 13(b) shows that a PE calculates eight MACs in parallel for eight input channels. The parallelism of the input channels is 64 (8 channels × 8 PEs). The IN memory *e* is not

Convolution

	Operations per cycle							
PS *A*(*E*)	R	W			R	W		
PS *B*(*F*)		R	W			R	W	
PS *C*(*G*)			R	W			R	W
PS *D*(*H*)				R	W			R

(a)

Deconvolution

	Operations per cycle							
PS *A*(*E*)	R	W	R	W	R	W	R	W
PS *B*(*F*)		R	W	R	W	R	W	R
PS *C*(*G*)	R	W	R	W	R	W	R	W
PS *D*(*H*)		R	W	R	W	R	W	R

(b)

Fig. 15 Role of the PS memory per cycle (R, read; W, write).

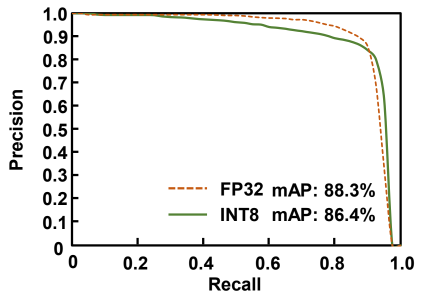


Fig. 17 Accuracy comparison in KITTI database: FP32 and INT8.

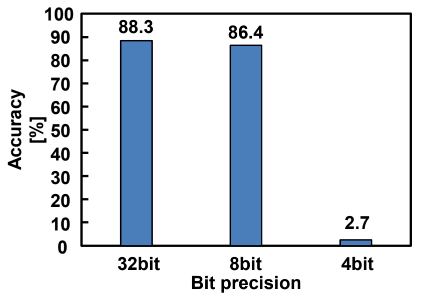


Fig. 18 Degradation in accuracy with bit precision.

used. The 64 channels at the same coordinate are divided into eight channels and are stored in the IN memories *a*–*d* and *f*–*i*. In this case, in the IN memory aligner, dummy data zero are fed for the IN memory *e*.

A 4 × 4 deconvolution operation with a stride of two extends a feature map by a factor of four, as presented in Fig. 14. The 4 × 4 deconvolution is well implemented, as presented in Fig. 13(c). The 2 × 2 partial sums are obtained by 3 × 3 inputs and a 4 × 4 kernel. Two PEs in different PECs are connected because 16 MACs are necessary for deconvolution. Some input data must be shared in the deconvolution. Therefore, they must be sent to the MAC via the selector. Their wiring lengths differ slightly from those in convolution, but no overhead on power exists because routing is implemented small multiplexers in a PE. Consequently, the parallelism of output channels is reduced to eight in the deconvolution.

C. *PS memory aligner*

Partial sum has a 16-bit width to suppress deterioration of the detection accuracy. The partial sums are read out from PS memories through the PS memory aligner. After the partial sums and activations are added up, new partial sums are written to the PS memories in the next cycle. A PS memory holds 64 bit width (8 bits × 8 output channels). Because a partial sum has 16-bit length, it must be divided into upper and lower parts: The

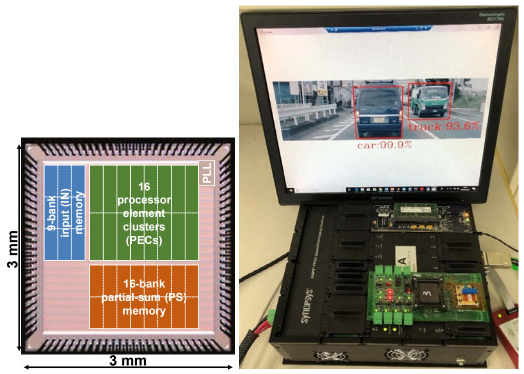

9-bank input (IN) memory
16 processor element clusters (PECs)
PLL
16-bank partial-sum (PS) memory
3 mm
3 mm
truck:93.6%
car:99.9%

Fig. 19 Chip micrograph and demonstration system.

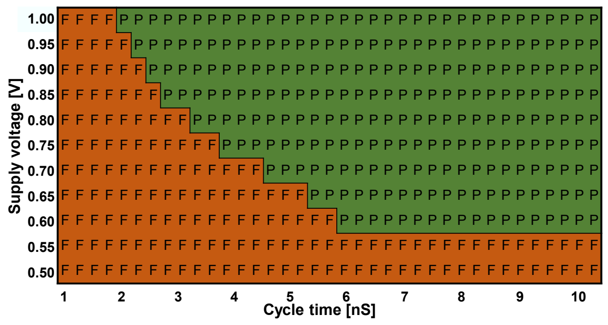


Fig. 20 Shmoo plot of test chip.

PS memories have a pair configuration (*A*0 and *A*1, *B*0 and *B*1, …, *H*0 and *H*1).

In the convolution, partial sums for PECs 0–7 are read out sequentially in either pair of PS memories *A*–*D*, as presented in Fig. 15(a). Similarly, partial sums for PECs 8–15 come from either pair of PS memories *E*–*H*. At the same time, renewed partial sums output from the accumulator 0 (ACC0) are written back to another pair of PS memories.

In the deconvolution, two partial sums exist from ACC1 and ACC2 in a PEC. Two pairs of PS memories *A*–*D* hold outputs for PECs 0–7 (Fig. 15(b): The same is applied to PS memories *E*–*H* for PECs 8–15). The internal memory bandwidth for the deconvolution is twice that used for the convolution.

VI. Measurement Results

Fig. 16 shows some detection examples on KITTI with FP32 Caffe simulation and proposed processor (INT8). The detection error is 1.9% compared to FP32 after retraining with Ristretto [40]. In some cases, the proposed processor detects objects that were not detectable in a single-precision floating point. Fig. 17 shows precision-recall curves for FP32 and INT8: their mean average precisions (mAPs) are 88.3% and 86.4%, respectively. As well, Fig. 18 shows the degradation of accuracy due to the bit precision. The accuracy in use of a 4-bit precision after retraining is illustrated as a reference; its accuracy is fatally degraded in object detection.

Fig. 19 portrays a chip micrograph and a demonstration system with a hosting FPGA. The test chip was fabricated in a TSMC 40-nm generic process. Fig. 20 is a Shmoo plot. The test chip operates at a supply voltage of 0.6–1.0 V and a frequency of 174–500 MHz (it does not function below 0.6 V because of

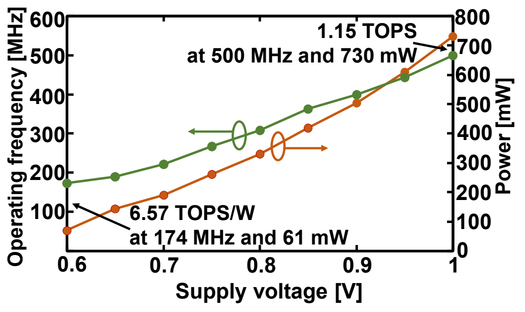


Fig. 21 Operation frequency and power characteristics.

TABLE II Simulated and measured power breakdowns to circuit components in the architecture at 500 MHz.

Component		Power [mW]	
		Simulation	Measurement
Logic circuitry	Clock	83.9	501
	Combinational logic	398.4	
Memory		210.7	229
Sum		693.0	730

the PLL). Fig. 21 shows the operating frequency and power characteristics. The maximum operating frequency of 500 MHz was verified at a supply voltage of 1 V; peak performance was achieved as 1.15 TOPS. The maximum energy efficiencies are 6.57 TOPS/W at 174 MHz and 0.6 V. TABLE II presents the simulated and measured power breakdowns to circuit components in the architecture. The simulation was conducted under typical process conditions and at room temperature. The powers in the simulation and the measurement match well.

The external memory bandwidth for DRAM shows strong effects on the inference time. Fig. 22 is a so-called roofline model [42], which shows a frame rate versus memory bandwidth for the target network. The proposed processor achieves up to 38.76 fps when the memory bandwidth is 20 GB/s.

TABLE III presents specifications that are useful for comparison among INT8 processors. One earlier study [21] exploited network sparsity, with energy-efficient and peak performance reaching 62.1 TOPS/W and 5.638 TOPS, respectively, when inputs and weights both have sparsity of 5%. In practical use, however, the energy efficiency is reduced drastically to 1.038 TOPS/W. Another study [23] examined adoption of 14-nm tri-gate technology by Intel Corp. Although it brings excellent energy efficiency of 11.3 TOPS/W, its peak performance is limited by the small chip area. Another study [27] achieved 6.9 TOPS and 11.5 TOPS/W with 75% zero weight. However, the performance and energy efficiency decreased respectively to 1.91 TOPS and 3.4 TOPS/W as the ratio of zero weight decreased. The proposed processor exhibits remarkable precision and figures for new applications such as automated driving of cars, which requires highly energy-efficient multi-scale object detection.

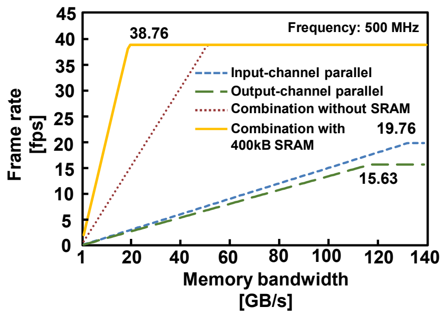


Fig. 22 Inference time by different parallel strategies.

VII. Summary

We described a 40-nm object detection processor for automated driving of cars. The processor performs only three operations: 3 × 3 convolution, 1 × 1 convolution, and 4 × 4 deconvolution. Multi-scale object detection is possible by virtue of the deconvolution feature. In the target network model, the bit widths of weights and activations are reduced to INT8 from the original FP32 Caffe model. Partial sums and batch normalization parameters are 16-bit precision to suppress the deterioration of accuracy. The input channel parallel and output channel parallel combination reduces the external memory bandwidth to 0.50 GB per KITTI-sized image with internal memory capacity of 400 kB. The detection error was found to be 1.9% of that produced by the original FP32 Caffe model. Its maximum operating frequency is 500 MHz at a supply voltage of 1 V. Its peak performance is 1.15 TOPS. Measurement results show that its maximum energy efficiency is 6.57 TOPS/W at 174 MHz and 0.6 V.

TABLE III Specifications of INT8 DNN processors

	[16]	[18]	[21]	[23]	[27]	[28]	This work
INT precision	8	8 (1, 4, 16)	8	8 (16, FP16)	8(16)	8	8
Purpose	CONV/FC/REC/ POOL/LSTM	CONV/FC/RNN	CONV/FC/ POOL	General purpose	CONV/FC	CONV	CONV/ DECONV
Process [nm]	65	65	65	14	8	28	40
Area [mm^2]	19.36	16	12	0.024	5.5	10.92	9
Supply voltage [V]	0.67–1.2	0.63–1.1	0.67–1.0	0.28–0.9	0.5–0.8	0.63–0.9	0.6–1.0
Frequency [MHz]	10–200	200	10–200	2.3–1460	67–933	90–215	174–500
Power [mW]	4–447	3.2–297	20.5–248.4	0.026–93.3	39–1553	61.75–243.6	61–730
Energy efficiency [TOPS/W]	5.09	5.57	1.038	11.3	3.4	5.34	6.57
Peak performance [TOPS]	0.4096	0.6912	5.638	0.1866	1.91	0.8796	1.15

CONV: convolution, FC: full connect, REC: recurrent
POOL: pooling, LSTM: Long short-term memory, DECONV: deconvolution

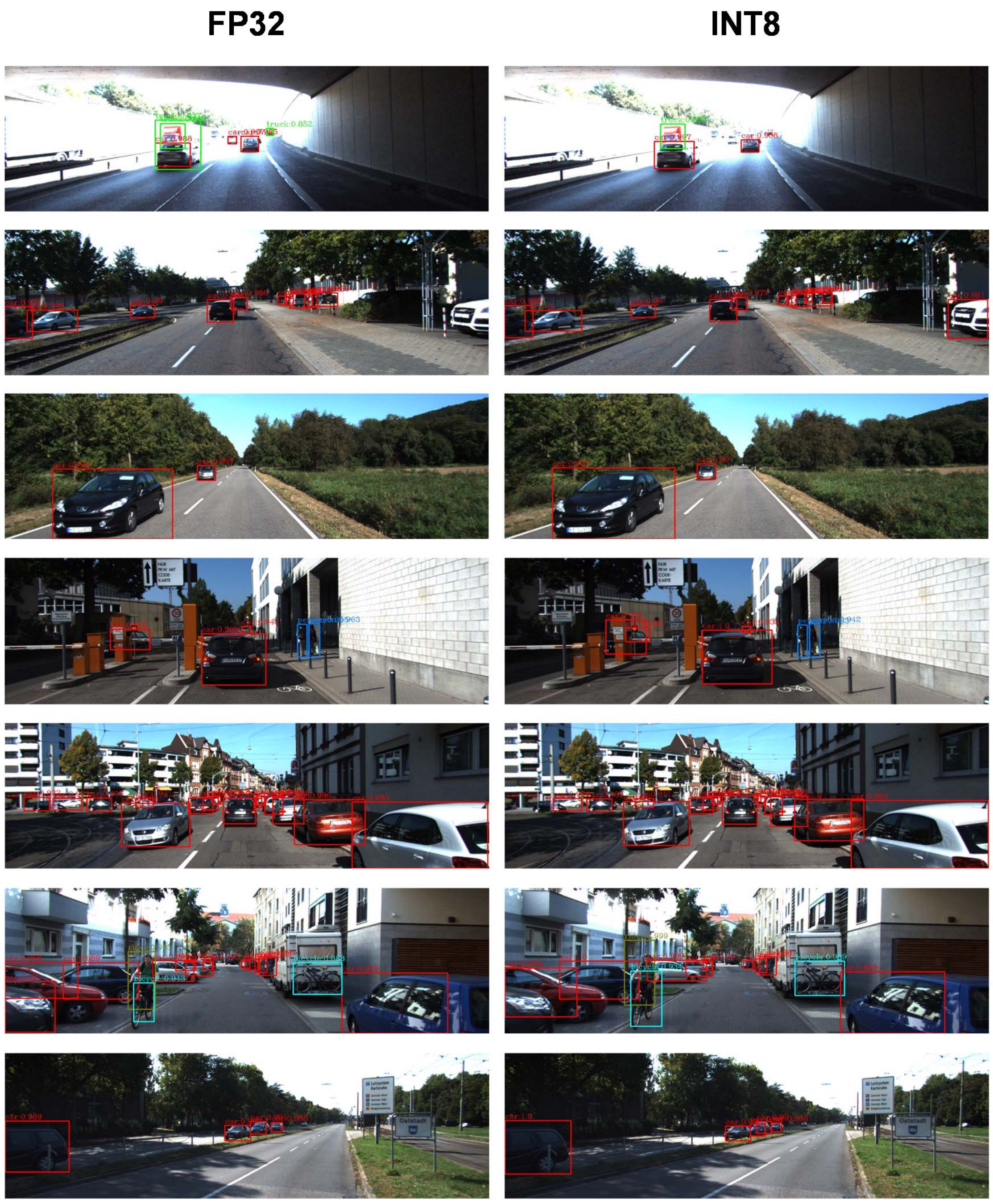


Fig. 16 Detection examples with Caffe FP32 and INT8 proposed processor.

REFERENCES

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Conference on Computer Vision and Pattern Recognition (CVPR), pp. 580–587, June 2014.

[2] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, "Selective Search for Object Recognition," International Journal of Computer Vision (IJCV), vol. 104, pp. 154–171, Sep. 2013.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," European Conference on Computer Vision (ECCV), pp. 346–361, Sep. 2014.

[4] R. Girshic, "Fast R-CNN," International Conference on Computer Vision (ICCV), pp. 1440–1448, Dec. 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Network," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, June 2017.

[6] J. Redmon, S. Divvala, R. Grishick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, June 2016.

[7] W. Liu, D. Anguelov, D. Erhan, S. Christian, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," European Conference on Computer Vision (ECCV), pp. 21–37, Oct. 2016.

[8] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. Berg, "DSSD: Deconvolutional Single Shot Detector," arXiv:1701.06659, Jan. 2017.

[9] Z. Li and F. Zhou, "FSSD: Feature Fusion Single Shot Multibox Detector," arXiv:1712,00960, Dec. 2017.

[10] 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," ACM/IEEE International Symposium on Computer Architecture (ISCA), pp. 243–254, June 2016.

[11] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[12] M. Horowitz, "Computing's energy problem (and what we can do about it)," International Solid-State Circuits Conference (ISSCC), pp. 10–14, Feb. 2014.

[13] D. Shin, J. Lee, J. Lee, and H. Yoo, "DNPU: An 8.1 TOPS/W Reconfigurable CNN-RNN Processor for General-Purpose Deep Neural Networks," International Solid-State Circuits Conference (ISSCC), pp. 240–241, Feb. 2017.

[14] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "ENVISION: A 0.26-to-10TOPS/W Subword-Parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28 nm FDSOI," International Solid-State Circuits Conference (ISSCC), pp. 246–247, Feb. 2017.

[15] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. T.-Yamazaki, T. Kuroda, and M. Motomura, "BRein Memory: A 13-Layer 4.2 K Neuron/0.8 M Synapse Binary/Ternary Reconfigurable in-Memory Deep Neural Network Accelerator in 65 nm CMOS," Symposia on VLSI Circuits, pp. C24–C25, June 2017.

[16] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 TOPS/W Reconfigurable Hybrid-Neural- Network Processor for Deep Learning Applications," Symposia on VLSI Circuits, pp. C26–C27, June 2017.

[17] K. Ueyoshi, K. Ando, K. Hirose, S. T.-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: A 7.49 TOPS Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96MB 3D SRAM Using Inductive-Coupling Technology in 40 nm CMOS," International Solid-State Circuits Conference (ISSCC), pp. 216–217, Feb. 2018.

[18] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "UNPU: A 50.6 TOPS/W Unified Deep Neural Network Accelerator with 1b-to16b Fully Variable Weight Bit-Precision," International Solid-State Circuits Conference (ISSCC), pp. 218–219, Feb. 2018.

[19] D. Bankman, L. Yang, B. Moons, and M. Verhelst, "An Always-On 3.8μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor with All Memory on Chip in 28 nm CMOS," International Solid-State Circuits Conference (ISSCC), pp. 222–223, Feb. 2018.

[20] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An Energy-Efficient SRAM with Embedded Convolution Computation for Low-Power CNN-Based Machine Learning Applications," International Solid-State Circuits Conference (ISSCC), pp. 488–489, Feb. 2018.

[21] Z. Yuan, J. Yue, H. Yang, Z. Wang, J. Li, Y. Yang, Q. Guo, X. Li, M.-F. Chang, H. Yang, and Y. Liu, "STICKER: A 0.41–62.1 TOPS/W 8bit Neural Network Processor with Multi-Sparsity-Compatible Convolution Arrays and Online Tuning Acceleration for Fully Connected Layers," Symposia on VLSI Circuits, pp. 33–34, June 2018.

[22] S. Yin, P. Ouyang, J. Yang, T. Lu, X. Li, L. Liu, and S. Wei, "An Ultra-High Energy-Efficient Reconfigurable Processor for Deep Neural Networks with Binary/Ternary Weights in 28 nm CMOS," Symposia on VLSI Circuits, pp. 37–38, June 2018.

[23] M. Anders, H. Kaul, S. Mathew, V. Suresh, S. Satpathy, A. Agarwal, S. Hsu, and R. Krishnamurthy, "2.9 TOPS/W Reconfigurable Dense/Sparse Matrix-Multiply Accelerator with Unified INT8/INT16/FP16 Datapath in 14 nm Tri-gate CMOS," Symposia on VLSI Circuits, pp. 39–40, June 2018.

[24] S. Kang, J. Lee, C. Kim, and H. Yoo, "B-Face: 0.2 mW CNN-Based Face Recognition Processor with Face Alignment for Mobile User Identification," Symposia on VLSI Circuits, pp. 137–138, June 2018.

[25] S. Yin, P. Ouyang, S. Zheng, D. Song, X. Li, L. Liu, and S. Wei, "A 141 uW, 2.46 pJ/Neuron Binarized Convolutional Neural Network based Self-Learning Speech Recognition Processor in 28 nm CMOS," Symposia on VLSI Circuits, pp. 139–140, June 2018.

[26] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A Mixed-Signal Binarized Convolutional-Neural-Network Accelerator Integrating Dense Weight Storage and Multiplication for Reduced Data Movement," Symposia on VLSI Circuits, pp. 141–142, June 2018.

[27] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "An 11.5TOPS/W 1024-MAC Butterfly Structure Deal-Core Sparsity-Aware Neural Processing Unit in 8 nm Flagship Mobile SoC," International Solid-State Circuits Conference (ISSCC), pp. 130–131, Feb. 2019.

[28] Z. Li, Y. Chen, L. Gong, L. Liu, D. Sylvester, D. Blaauw, and H.-S. Kim, "An 879GOPS 243 mW 80 fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration," International Solid-State Circuits Conference (ISSCC), pp. 134–135, Feb. 2019.

[29] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 696–701, June 2014.

[30] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," ACM/IEEE International Symposium on Computer Architecture (ISCA), pp. 92–104, June 2015.

[31] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," ACM/IEEE International Symposium on Computer Architecture (ISCA), pp. 367–379, June 2016.

[32] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv: 1502.03167, Feb. 2015.

[33] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 315–323, Apr. 2011.

[34] KITTI: www.cvlibs.net/datasets/kitti.

[35] ImageNet: www.image-net.org.

[36] Caffe: https://caffe.berkeleyvision.org.

[37] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556, Sep. 2014.

[38] H. Naganuma and R. Yokota, "Accelerating Convolutional Neural Networks Using Low Precision Arithmetic," International Conference on High Performance Computing in Asia–Pacific Region (HPCAsia), 2018.

[39] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 26–35, Feb. 2016.

[40] Ristretto: http://lepsucd.com/ristretto-cnn-approximation.

[41] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "ThUnderVolt: Enabling Aggressive Voltage Underscaling and Timing Error Resilience for Energy Efficient Deep Learning Accelerators," ACM Design Automation Conference (DAC), no. 19, June 2018.

[42] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 161–170, Feb. 2015.

Reiya Kawamoto received a B.E. degree in Computer Science and System Engineering from Kobe University, Kobe, Japan in 2018. He is currently in the master course at Kobe University. His current research is a low-power Deep Learning processor for automotive systems.

Masakazu Taichi received a B.E. degree in Computer Science and System Engineering from Kobe University, Kobe, Japan in 2018. He is currently in the master course at Kobe University. His current research is a low-power Deep Learning processor for automotive systems..

Masaya Kabuto received a B.E. degree in Computer Science and System Engineering from Kobe University, Kobe, Japan in 2018. He is currently in the master course at Kobe University. His current research is a low-power Deep Learning processor for automotive systems.

Daisuke Watanabe received a B.E. degree in Computer Science and System Engineering from Kobe University, Kobe, Japan in 2018. He is currently in the master course at Kobe University. His current research is a low-power Deep Learning processor for automotive systems.

Shintaro Izumi received his B.Eng. and M.Eng. degrees in Computer Science and Systems Engineering from Kobe University, Hyogo, Japan, respectively, in 2007 and 2008. He received his Ph.D. degree in Engineering from Kobe University in 2011. He was a JSPS research fellow at Kobe University from 2009 to 2011. Since 2011, he has been an Assistant Professor in the Organization of Advanced Science and Technology at Kobe University. His current research interests include biomedical signal processing, communication protocols, low-power VLSI design, and sensor networks.

He has served as a Vice Chair of IEEE Kansai Section Young Professional Affinity Group, as a Student Activity Committee Member for IEEE Kansai Section, as a Program Committee Member for IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips), and as a Guest Associate Editor of IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences. He was a recipient of the 2010 IEEE SSCS Japan Chapter Young Researchers Award.

Masahiko Yoshimoto joined the LSI Laboratory, Mitsubishi Electric Corporation, Itami, Japan, in 1977. From 1978 to 1983 he was engaged in the design of NMOS and CMOS static RAM. Since 1984, he was involved in the research and development of multimedia ULSI systems. He earned a Ph.D. degree in Electrical Engineering from Nagoya University, Nagoya, Japan in 1998. Since 2000, he was a professor of Dept. of Electrical & Electronic System Engineering in Kanazawa University, Japan. Since 2004, he has been a professor of the Dept. of Computer and Systems Engineering in Kobe University, Japan. His current activity specifically emphasizes the research and development of ultra-low-power multimedia and ubiquitous media VLSI systems and a dependable SRAM circuit. He holds 70 registered patents. He served on the program committee of the IEEE International Solid State Circuit Conference from 1991 to 1993. In addition, he served as Guest Editor for special issues on Low-Power System LSI, IP and Related Technologies of IEICE Transactions in 2004. He was a chair of IEEE Solid State Circuits Society (SSCS) Kansai Chapter from 2009 through 2010. He is also a chair of the IEICE Electronics Society Technical Committee on Integrated Circuits and Devices from 2011–2012. He received R&D100 awards from R&D magazine for the development of the DISP and the development of the real-time MPEG2 video encoder chipset respectively in 1990 and 1996. He also received the 21st TELECOM System Technology Award in 2006.

Hiroshi Kawaguchi received B.Eng. and M.Eng. degrees in electronic engineering from Chiba University, Chiba, Japan, in 1991 and 1993, respectively, and earned a Ph.D. degree in electronic engineering from The University of Tokyo, Tokyo, Japan, in 2006.

He joined Konami Corporation, Kobe, Japan, in 1993, where he developed arcade entertainment systems. He moved to The Institute of Industrial Science, The University of Tokyo, as a Technical Associate in 1996, and was appointed as a Research Associate in 2003. In 2005, he moved to The Graduate School of Engineering, Kobe University, Kobe, Japan, as a Research Associate. From 2015 to 2016, he was a Visiting Researcher at Politecnico di Milano. Since 2016, he has been a Full Professor at The Graduate School of Science, Technology and Innovation, Kobe University. He is also a Collaborative Researcher with The Institute of Industrial Science, The University of Tokyo. His current research interests include low-voltage operating circuits, soft error characterization and mitigation, ubiquitous sensor networks, organic semiconductor circuits, data converters, healthcare devices, and neuro computer architecture.

Dr. Kawaguchi was a recipient of IEEE ISSCC 2004 Takuo Sugano Outstanding Paper Award, ACM/IEEE ASP-DAC 2013 University Design Contest Best Design Award, and IEEE ICECS 2016 Best Paper Award. He has served as a Design and Implementation of Signal Processing Systems (DISPS) Technical Committee Member for IEEE Signal Processing Society, as a Technical Program Committee Member for IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE Global Conference on Signal and Information Processing (GlobalSIP), IEEE Workshop on Signal Processing Systems (SiPS), IEEE Custom Integrated Circuits Conference (CICC), and IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips), as an Organizing Committee Member for IEEE Asian Solid-State Circuits Conference (A-SSCC), and ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), and as an Associate Editor of Springer Journal of Signal Processing Systems, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, IEICE

Transactions on Electronics, and IPSJ Transactions on System LSI Design Methodology (TSLDM).He is a member of the IEEE, ACM, and the IEICE.

Go Matsukawa received Bachelor's and Master's in Computer and Systems Engineering from Kobe University in 2013 and 2014. He received his Ph.D. degree in Engineering from Kobe University in 2017. Since 2018, he has been with Toyota Motor Corporation, Toyota City, Japan, where he is working on Toyota Research Institute Advanced Development. He is currently engaged in the development of SoC and High Performance Computing system for automated driving.

Toshio Goto received Bachelor's and Master's degree in Electronic Engineering from Meiji University in 1997 and 1999, respectively. Since 1999, he has been with Toyota Motor Corporation, Toyota City, Japan, where he is working on Toyota Research Institute Advanced Development. He is currently engaged in the development of SoC and High Performance Computing system for automated driving.

Motoshi Kojima received a Bachelor's degree in Computer Science from The University of Electro-Communications in 1989. Since 1989, he has been with Toyota Motor Corporation, Japan. He is currently a Lead of High-Performance Computing at Toyota Research Institute Advanced Development, Inc.