A 12.1 TOPS/W Quantized Network Acceleration Processor With Effective-Weight-Based Convolution and Error-Compensation-Based Prediction

Huiyu Mo, Wenping Zhu, *Member, IEEE*, Wenjing Hu, Qiang Li, Ang Lieps, Shouyi Yin, *Member, IEEE*, Shaojun Wei, *Fellow, IEEE*, and Leibo Liu, *Senior Member, IEEE*

***Abstract*— In this article, a quantized network acceleration processor (QNAP) is proposed to efficiently accelerate CNN processing by eliminating most unessential operations based on algorithm-hardware co-optimizations. First, an effective-weight-based convolution (EWC) is proposed to distinguish a group of effective weights (EWs) to replace the other unique weights. Therefore, the input activations corresponding to the same EW can be accumulated first and then multiplied by the EW to reduce amounts of multiplication operations, which is efficiently supported by the dedicated process elements in QNAP. The experimental results show that energy efficiency is improved by 1.59×–3.20× compared with different UCNN implementations. Second, an error-compensation-based prediction (ECP) method adopts trained compensated values to replace partly unimportant partial sums to further reduce potentially redundant addition operations caused by the ReLU function. Compared with SnaPEA and Pred on AlexNet, 1.23× and 1.75× higher energy efficiencies (TOPS/W) are achieved by ECP, respectively, with marginal accuracy loss. Third, the residual pipeline mode is proposed to efficiently implement residual blocks with a 1.5× lower memory footprint, a 1.18× lower power consumption, and a 13.15% higher hardware utilization on average than existing works. Finally, the QNAP processor is fabricated in the TSMC 28-nm CMOS process with a core area of 1.9 mm^2. Benchmarked with AlexNet, VGGNet, GoogLeNet, and ResNet on ImageNet at 470 MHz and 0.9 V, the processor achieves 117.4 frames per second with 131.6-mW power consumption on average, which outperforms the state-of-the-art processors by 1.77×–24.20× in energy efficiency.**

***Index Terms*— Convolutional neural network (CNN), pipeline mode, prediction, weight decomposition.**

Manuscript received April 13, 2021; revised June 27, 2021, August 13, 2021, and September 5, 2021; accepted September 14, 2021. Date of publication October 13, 2021; date of current version April 25, 2022. This article was approved by Associate Editor Dennis Sylvester. This work was supported in part by the National Natural Science Foundation of China under Grant 61834002 and in part by the National Key Research and Development Program of China under Grant 2018YFB2202101. *(Huiyu Mo and Wenping Zhu contributed equally to this work.) (Corresponding author: Leibo Liu.)*

Huiyu Mo, Wenjing Hu, Ang Li, Shouyi Yin, Shaojun Wei, and Leibo Liu are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China (e-mail: liulb@tsinghua.edu.cn).

Wenping Zhu is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China, and also with the Institute of Semiconductors, Chinese Academy of Sciences, Beijing 100083, China.

Qiang Li is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China, and also with Intel Corporation, Beijing 100080, China.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2021.3113569.

Digital Object Identifier 10.1109/JSSC.2021.3113569

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have achieved great success in computer vision [1], natural language processing [2], and speech recognition [3]. However, these works largely rely on billions of computations to achieve high accuracy. For example, VGGNet [4] consumes nearly 46.1 billion multiply–accumulate (MAC) operations to make great breakthroughs in image classification, which normally runs on modern GPUs with very high computational capability. A large number of computations require too many hardware resources to maintain acceptable accuracy and running speed, which incurs unbearable area and power overheads [5]–[7].

To alleviate these overheads, a number of processors have been designed to accelerate quantized CNN processing. Most of them focus on architecture design to decrease data communication [8], [9] or use low-power design approaches to reduce power consumption [10], [11]. In contrast, some other processors [12]–[15] make use of weight sparsity and redundancy to implement computations of only nonzero weights and activations to accelerate the quantized CNN process. However, a large number of redundant multiplications caused by repetitive quantized weights in each kernel of CNNs are not considered in these processors. Meanwhile, many convolutional (CONV) operations that produce negative activations are actually unnecessary if the ReLU function is used, which is also not fully investigated in current CNN processors. Furthermore, most CNN processors implement the CONV process based on the layer-by-layer mode, which requires an amount of off-chip memory access, especially in skip-connection-based modules. As off-chip memory access consumes much higher power than MAC [8], some processors adopt the pipeline mode to alleviate this problem [6], [16]. However, low hardware utilization overhead is caused in these processors.

Witnessing the huge potential enabled by repetitive weights, effective-weight-based convolution (EWC) is first proposed to select a group of effective weights (EWs) to replace the other unique weights in each kernel through decomposition. The resource requirement is reduced substantially relative to unlimited unique weights in UCNN [17]. Then, the multiplication associative law is used in EWC to reduce multiplications by an average of 9.1× compared to UCNN [17]. Second, error-compensation-based prediction (ECP) is proposed to

decompose CONV operations into activation high-order bits (I_{HBs}) operations and low-order bits (I_{LBs}) operations. When the partial sum (PSum) produced by I_{HBs} operations is smaller than the pretrained threshold, the remaining I_{LBs} operations will be abandoned to accelerate CNN implementations, as negative activations will be set to zero in the ReLU function. In particular, when only a few activations require I_{LBs} operations in one cycle and incur a large waste of hardware resource, pretrained compensation values are used directly to compensate these I_{LBs} operations without consuming cycles, achieving at least 75% hardware utilization and obtaining a negligible accuracy loss. Third, as more off-chip memory access occurs in skip-connection-based blocks, such as the residual block [18] and inception block [19], a residual pipeline (RP) mode is designed with all hardware resources assigned for one layer at one period, which achieves higher hardware utilization than previous pipeline modes [6], [16], [20]. Here, skip connection-based blocks are called residual blocks for simplification. Finally, a quantized network acceleration processor, namely, QNAP, is designed in accordance with the above techniques to implement mainstream CNN models efficiently.

The main contributions of QNAP are given as follows.

1) EWC is proposed to avoid a large number of multiplications caused by the repetition of quantized weights. Hardware-friendly scheduling is further proposed to address irregular CONV operations.
2) ECP is proposed to eliminate unnecessary I_{LBs} CONV operations caused by the ReLU function. Furthermore, a specific compensation strategy is designed to use values to further accelerate CNN implementation and improve hardware utilization with ignorable accuracy loss.
3) The RP mode is proposed to support residual blocks with less off-chip memory access, which achieves higher hardware utilization than previous pipeline modes.
4) An energy-efficient architecture is designed to support mainstream CNN models with the proposed techniques used. In particular, the architecture can efficiently avoid the complicated control logic caused by irregular weight distribution.

Compared to prior work published in ISSCC'21 [21], this article provides more detailed interpretations of the proposed methods and more comprehensive evaluations and comparisons with state-of-the-art (SOTA) solutions. Furthermore, the feasibility and deployment mechanism to support various models, including different structures and kernel sizes, is first discussed in this work.

QNAP is fabricated in a 28-nm TSMC process. The processor can support 1-8-b quantized weights of CNNs and consumes 206 kB on-chip SRAM. Including the evaluated benchmark networks (AlexNet [22], VGGNet [4], ResNet (ResNet-50 is used here) [18], and GoogLeNet [19]), the processor can be programmed based on the preset configuration information to support CNNs with different kernel sizes, activation functions, strides, and so on. The implementation results demonstrate the flexibility and scalability of the architecture. In addition, the processor achieves high effective energy and area efficiency based on the proposed techniques compared with the SOTA CNN processors. Benchmarked with benchmark networks on ImageNet [22], the highest energy efficiency of 12.62 TOPS/W (on ResNet), and average effective energy efficiency of 12.1 TOPS/W, lowest frame energy of 0.46 mJ/Frame (on AlexNet) and effective area efficiency of 745.1 GOPS/mm^2 are achieved at 470 MHz and 0.9 V. Furthermore, the frame energy of QNAP is at most 89.77$\times$ lower than that of SOTA processors (analyzed in Section VII-D).

The remainder of this article is organized as follows. Section II reviews related research and introduces the design considerations. Three proposed techniques are described in Sections III–V. Section VI introduces the system architecture of the proposed processor. In Section VII, chip implementation and extensive measurement results are presented. Finally, Section VIII states the conclusion.

II. Motivation and Design Consideration

The main bottleneck of CNN processors stems from MACs and memory access in CNN inferences [8], [9], [14]. The number of MACs determines the performance of the processors, and memory access, especially off-chip memory access, significantly affects the overall power consumption. Therefore, the main contributions of the QNAP aim to alleviate the above overhead in order, including EWC, ECP for MAC reduction, and RP for off-chip memory access alleviation. The motivation of the three contributions is described as follows.

A. Motivation of EWC

The number of quantized weights is always limited due to the fixed bit width. For example, there are at most 256 unique weights for 8-b quantization in one kernel. Therefore, many repetitive quantized weights occur in each kernel, especially in the deeper layer with more channels. UCNN [17] exploits the above property by reusing subcomputations among the same feature map and all kernels. However, there are only a small number of identical sub-computations, which limits the total operation reduction. Furthermore, the number of unique weights varies greatly among different CNNs. In this case, maximal hardware resources should be assigned to support these computations, which leads to severe underutilization of hardware resources. In fact, the weight repetition rate in all channels of each kernel is much higher (more than 10$\times$) than that in the same channel of all kernels. Furthermore, weights in each kernel are found to approximately follow the normal distribution located in a limited range, which makes it possible to distinguish several basic values to cover all the weights in each kernel.

B. Motivation of ECP

To further reduce the computing complexity posed by a large number of additional operations in ReLU-involved networks, the prediction-based mechanism was introduced in [25] and [26] to terminate the calculation in advance if

the current result indicated that the final result was probably negative. Although impressive computations are reduced in these processors, large accuracy loss and complex control logic are required to maintain high performance. Inspired by the study [24], activations are first split into two parts, I_{HBs} and I_{LBs}. I_{HBs} are first processed, and the results are used to decide whether to abandon I_{LBs} computations. Although a high speedup can be achieved theoretically, a serious workload imbalance will be caused since I_{LBs} computations are uncertain for each activation. A complicated schedule method was proposed to accelerate CNN inference with higher hardware utilization in [24]. However, it incurs a large power consumption overhead. We find that lower hardware utilization is always caused by a small proportion of activations that require I_{LBs} computations. Meanwhile, they usually have a minor effect on the final accuracy. Therefore, these operations could be further avoided for higher speedup and hardware utilization.

C. Motivation of RP

In conventional layer-by-layer mode, all activations of each layer will be first transferred into off-chip memory and then read from off-chip memory for the next layer. However, activations in some layers of the residual block would be loaded repeatedly due to the skip connection, which is utilized in mainstream CNN models. Heavier memory traffic leads to higher power consumption. Considering that the off-chip memory access operation causes longer latency and higher power consumption than MACs (approximately 128×) [8], it should be reduced as much as possible. To address this issue, the pipeline mode has been recently widely studied [6], [16], [20] to reduce off-chip memory access by making multiple layers work simultaneously. In this way, the intermediate activations produced by each layer will be consumed by the subsequent layer immediately instead of being transferred to off-chip memory. However, as several CONV layers are always processed simultaneously, each CONV layer requires a different number of hardware resources to maintain high hardware utilization. Furthermore, as different CNNs are formed of various CONV layers, a fixed number of hardware resources cannot always ensure high hardware utilization (always lower than 90%). Therefore, improving higher hardware utilization and performance in pipeline mode is a problem.

III. Effective-Weight-Based Convolution

There are amount of quantized weights in quantized CNN models. However, the value of quantized weights is constrained by the quantization bitwidth. For example, there are 576 weights but at most 256 unique weights in an 8-b $3 \times 3 \times 64$ kernel, causing a 55.6% weight repetition rate. In fact, the weight repetition rate in each kernel of CNN models is much higher than that in this extreme case. Furthermore, repetitive quantized weights will cause redundant multiplications. In a conventional 3×3 convolution operation, each activation will be multiplied by the corresponding weight, as shown in Fig. 1(a), namely, nine multiplications in total. However, according to the multiplication associative law, the activations corresponding to the same weight can be first summed up and then multiplied by the weight only once,

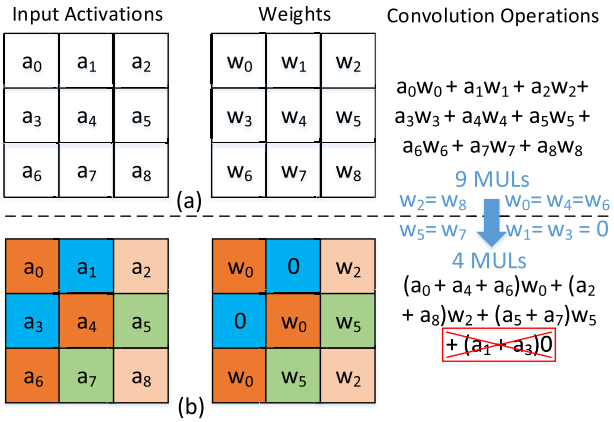


Fig. 1. Redundant multiplications caused by the repetitive quantized weights. (a) Conventional CONV process. (b) CONV process based on multiplication associative law.

as shown in Fig. 1(b). In this case, only four multiplications are required, with 55.6% of redundant multiplications eliminated. Furthermore, as the multiplications and additions of unique weights are implemented independently, these operations of zero-value weight can be eliminated more easily compared with most weight-sparsity-based methods.

To eliminate the redundant multiplications caused by the repetitive quantized weights, unique weights in each kernel are first screened out, and an associative law of multiplication is then used in our work, as shown in Fig. 1. However, the difference in the number of unique weights in each kernel increases the design complexity, especially for efficient resource allocation.

To address this problem, the distribution of unique weights is first investigated. Without loss of generality, AlexNet is taken as the example here, and the results are shown in Fig. 2. Three properties can be learned from the results. First, except for the first CONV layer kernels, the unique weights of the other CONV layer kernels approximately follow a Gaussian distribution. The reason for the irregular distribution of the weights of the first CONV layer might be caused by direct processing on raw image pixels, with the aim of selecting important objects and filtering out unnecessary information. In contrast, kernels in other layers focus on smaller attention scopes, processing regular Ifmaps and producing more regular Ofmaps. Second, the unique weights of each kernel in all CONV layers are almost symmetric on the Y-axis. This is largely because the property of symmetry always occurs in most objects. Therefore, CNN models always detect particular patterns at multiple spatial locations in the input, and these patterns often occur in different orientations [25]–[27]. Third, the values of unique weights in each kernel are within a limited range. The reason is that the function of each CONV layer is different, and kernels in the same CONV layer also focus on different aspects [28]. Namely, a group of centralized weights is required to realize this goal. These properties are also found in other benchmark CNN models based on the experimental results.

Based on the above analysis, EWC is proposed to effectively minimize the number of unique weights, as shown in Fig. 3(a). Given all unique weights $\langle w_0, w_1, \ldots, w_7 \rangle$,

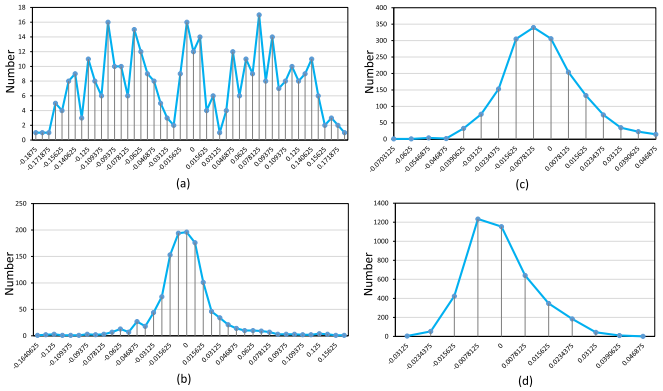


Fig. 2. Unique weight distribution in AlexNet. (a) First kernel of CONV1. (b) First kernel of CONV2. (c) First kernel of CONV4. (d) First kernel of FC7.

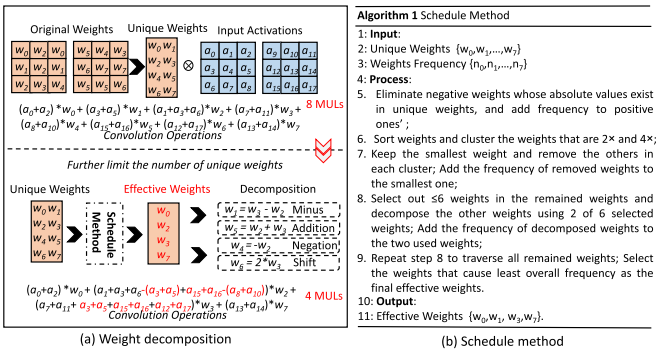


Fig. 3. (a) Principle of EWC. (b) Schedule method.

the useless zero-value weight is first abandoned. Then, according to the distribution property of unique weights, a schedule method is proposed in EWC to select EWs $\langle w_0, w_2, w_3, w_7\rangle$. The EWs are used to decompose the other unique weights based on addition, minus, negation, and shift operations, such as $w_1 = w_3 - w_2$ and $w_4 = 2 * w_2$. The activations of decomposed unique weights are assigned to the corresponding EWs without accuracy loss. Based on EWC, the number of multiplications can be further reduced. As shown in Fig. 3(a), 50% of multiplications can be reduced. Although 25% extra addition operations are brought in this case, this overhead can be effectively kept less than 1% for most models with the adoption of the schedule method in EWC. Note that EWC is still applicable to the first layer, whose unique weights do not follow a Gaussian distribution. In addition, the additional overhead is also negligible as the first layer always takes a little percentage of the overall computation. For instance, this overhead is only 1.4% for ResNet.

As described in Fig. 3(b), the schedule method is proposed to ensure that all unique weights can be decomposed with the EWs completely at a minimum cost. In addition to the value of unique weights, their frequency $\langle n_0, n_1, \ldots, n_7\rangle$ should be provided. First, according to the symmetry property, the negative weights, absolute values of which belong to the unique weights, are eliminated. Meanwhile, the frequency of these negative weights is added to that of their corresponding positive weights. Assuming that $w_4 = -w_2$, w_4 will be abandoned, and its frequency is added to that of w_2, $n_2 = n_2 + n_4$. In this way, the sums of activations corresponding to the eliminated negative weights can be added to that of their positive weights only through negation operation. This can nearly eliminate half of the unique weights, as shown in Fig. 2. Then, based on the clustering results, all unique weights are first sorted, and then, unique weights that are $2\times$ or $4\times$ of each other are clustered. For example, as $w_6 = 2 * w_3$, w_6 will be eliminated, and $n_3 = n_3 + n_6$. Namely, the activation of w_6 only requires left shift and then can be added to activations of w_3. Furthermore, in addition to the above single negation, $2\times$ shift, $4\times$ shift operations, and addition/minus operations of two EWs based on the combination of these single operations are used to further decompose unique weights. Note that only 1- and 2-b left-shifting operations (namely, $2\times$ and $4\times$) are considered in this work, as the edge unique weights are always close to the central unique weights. Furthermore, no more than six EWs are selected from the remaining unique weights, and the other weights will be further decomposed by two EWs. The number of EWs, six, is obtained based on massive experiments, which shows that unique weights in most 8-bit-quantization kernels can be completely covered by six EWs. For example, as $w_1 = w_3 - w_2$, w_1 will be eliminated, and the corresponding frequency will be updated, $n_3 = n_3 + n_1$ and $n_2 = n_2 + n_1$. However, the activations of w_1 have to be added to both that of w_2 and w_3, which causes extra addition operations. Considering the Gaussian distribution, different unique weight decompositions will cause different numbers of extra addition operations. The principle of EW selection is that the decomposed unique weights should incur the fewest extra addition operations. To achieve this target, EWs should be selected from the unique weights that are distributed near the center of the Gaussian distribution. The unique weights that are distributed at the edge of the Gaussian distribution are preferred candidates to be decomposed. The reason is that central unique weights occupy more activations. If these unique weights are decomposed, potentially more addition operations will occur. The six EWs will be repeatedly chosen from the remaining unique weights in our work. Therefore, EWs are searched from the central unique weights to edge unique weights until six EWs are selected. In addition, if the number of unique weights after step 8 in Fig. 3(b) is less than six, the process will stop, and these unique weights will be directly used as the final EWs. Therefore, no more than six EWs will be obtained in the schedule method in most cases. Moreover, there may be more than six EWs in some specific layers, mainly the first CONV layer in most CNN models. This problem will be addressed in our hardware implementation.

The implementation process of EWC in the QNAP architecture can be described in four steps, as shown in Fig. 4. First, four successive weights from different channels are first chosen [e.g., $\langle w_0, w_6, w_7, w_8\rangle$, as shown in Fig. 4(a)]. Second, six EWs, w_0, w_1, w_2, w_3, w_4, and w_5, are used to decompose the four weights, namely, $w_0 = w_0$, $w_6 = -w_2$, $w_7 = 2 * w_2$, and $w_8 = w_3 - w_2$. Third, to efficiently support EWC in the QNAP architecture, original weights should be encoded as weight information (weight_info) based on the distribution of EWs in each kernel. As weight_info is encoded offline, no extra operations or delay will be consumed by the encoding

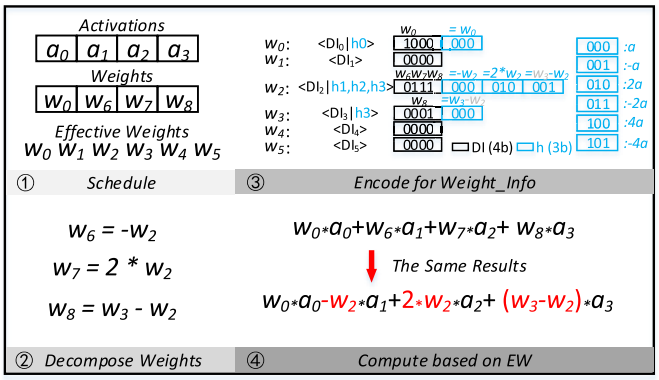


Fig. 4. Off-line weight encoding process.

process online. To form more hardware-friendly weight_info, six groups of codes ($\langle \mathrm{DI}_K | h0, \ldots, hk \rangle_{K=0,1,\ldots,5;k\leq 3}$) related to six EWs ($\langle w_0, \ldots, w_5 \rangle$) are set in our weight_info. 4-b DI_K indicates the indexes of input activations that relate to the Kth EW. For example, as shown in Fig. 4(a), $\mathrm{DI}_2 = 0111$ means that the decomposed weights $\langle w_6, w_7, w_8 \rangle$ are all related to EW w_2. Therefore, their corresponding activations $\langle a_1, a_2, a_3 \rangle$ should be assigned to the EW w_2. Furthermore, as EWs should be preprocessed to represent the original weights (e.g., w_2 should be negated to replace w_6), this relation is transferred to the activations to obtain the same convolution results as the original computations. For example, activation a_1 corresponding to w_6 should be negated and then multiplied by the w_2. To achieve this goal, 3-b hk is set in the weight_info, which represents the operation that should be implemented for the corresponding activation. The detailed operations represented by hk are shown in Fig. 4(a). The number of hk in each group of code is decided by the number of activations indexed by DI_K. For example, as $\langle w_6, w_7, w_8 \rangle$ are related to EW w_2, their activations are indexed by DI_2. Therefore, there are three hk in this group of code ($h1, h2$, and $h3$) to represent the operations that should be implemented in the activations. Note that there must be six DI codes ($\langle \mathrm{DI}_K \rangle_{K=0,1,\ldots,5}$) in a set of weight_info, even if no input activation is related to some EWs, and their corresponding DI will be assigned 0000. Namely, 24 bits can be consumed by six DI. Meanwhile, as each input data can possibly be decomposed by two EWs, namely, one input activation can mostly be assigned to two EWs, four input activations require eight hk codes (24 bits) at most. Therefore, a set of weight_info consumes 48 bits in total, representing 4 original weights. Fourth, weight_info can be used to obtain the same CONV results as the original CONV process. For example, a_3 will be assigned to w_2 and w_3 based on the DI_2 and DI_3 in weight_info. Meanwhile, according to the corresponding h, a_3 is negated for w_2 and unchanged for w_3. In this way, the proposed EWC can significantly reduce the overall multiplication operations without accuracy loss.

EWC is mainly designed for 3×3 and 1×1 kernels for two reasons. The first is that these large kernels are rarely used in mainstream networks, as they can only achieve marginal accuracy improvement with much more area consumption and lower hardware utilization. The second is that larger kernels may contain more than six EWs. To tackle the above problem, the kernel partition method [29] is adopted in QNAP. Taking the first layer (7×7 kernels with stride 2) in ResNet as an example, this 7×7 kernel will be first padded with zero to form an 8×8 kernel and then partitioned into 16 2×2 sub-kernels. Then, these subkernels are further padded with zero to form 3×3 subkernels. In our experiments, with the method [29] adopted for large kernels of four benchmark networks, the number of EWs in subkernels can be maintained at less than six. Even if sub-kernels of certain layers require more than six EWs after partitioning, two alternative methods could be utilized to ensure that QNAP can implement these networks end to end. The first is that large kernels are replaced by two 3×3 kernels. For example, the first layer (11×11 kernels with stride 2) of AlexNet is replaced with two 3×3 layers with stride 2 without accuracy loss. The second is that, based on the off-line weight encoding, the configuration information could configure QNAP to process this layer repeatedly to ensure that all EWs are used. As this extreme case never occurs in our experiments, we believe the probability that this situation occurs in other unevaluated networks is very low.

TABLE I
REDUCED RATIO OF MULTIPLICATION OPERATIONS

	AlexNet	VGGNet	GoogLeNet	ResNet	Average
CONV	96.8%	92.5%	92.3%	93.1%	93.7%
CONV +abs	98.1% (**+1.3%**)	93.7% (**+1.4%**)	94.9% (**+2.6%**)	95.6% (**+2.5%**)	95.6% (**+2.0%**)
CONV +abs+DP	99.7% (**+1.6%**)	98.9% (**+5.2%**)	99.1% (**+4.2%**)	99.2% (**+3.2%**)	99.2% (**+3.6%**)
FC	99.8%	99.8%	-	98.1%	99.2%
FC+abs	99.9%	99.9%	-	98.7%	99.5%
FC+abs +DP	99.9%	99.9%	-	99.6%	99.8%

The advantages brought by EWC on four benchmark networks, AlexNet [22], VGGNet [4], GoogLeNet [19], and ResNet-50 (called ResNet here) [18], are listed in Table I. EWC first directly eliminates repetitive quantized weights, reducing an average of 93.7% multiplications in CONV layers. Then, an absolute operation (abs) is used to remove negative weights whose absolute values exist in the present quantized weights, with a further 2.0% multiplication reduction. Finally, EWC adopts the decomposition technique (DP) to further constrain the number of quantized weights to avoid a serious waste of hardware resources, further reducing multiplications by 3.6% on average. As the FC layer is parameter intensive, EWC reduces the corresponding multiplications by 99.8% on average. The results show that the proposed EWC can eliminate most multiplications in quantized CNN models, which significantly benefits low-power hardware design.

IV. Error-Compensation-Based Prediction Method

The ReLU function is widely used in mainstream CNNs to improve the nonlinearity of the extracted features, which converts negative results to zero, as shown in Fig. 5(a). If the values of the convolutional results are negative, regardless of whether they are $-\infty$ or -0, the activations will be set to zero. In fact, in the number of convolutional computations, part of the convolution computations can decide the sign of the final results. Namely, if the sign of the result is negative,

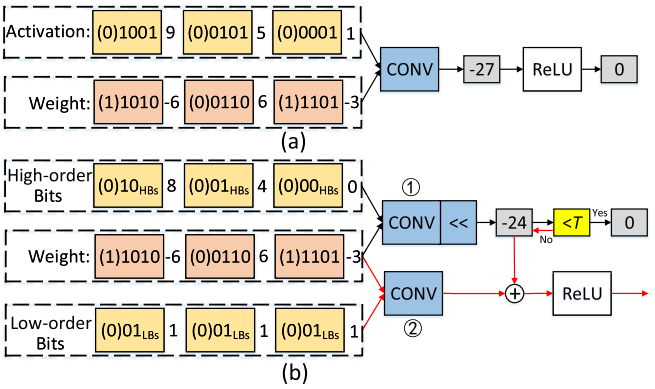


Fig. 5. (a) Conventional CONV process. (b) Prediction-based convolution process to reduce operations; T means the threshold.

the remaining computations can be abandoned, and instead, zero can be directly output, which can theoretically improve the performance of CNN implementation. Method [24] splits activations into I_{HBs} and I_{LBs} and uses I_{HBs} results to predict the sign of the final results, as shown in Fig. 5(b). If the I_{HBs} result is smaller than the pretrained threshold T (0 in [24]), 0 is output as the final activation. Otherwise, I_{LBs} will be convolved with weights, and the result is added with the result of I_{HBs}. However, more operations can be saved if the appropriate T is chosen. Furthermore, the extremely complicated control logic is used in [24] to achieve high hardware utilization with additional high power consumption overhead.

To address the problems in [24], ECP is proposed in this section. Meanwhile, as the workload of multiplications is reduced significantly by EWC, the main overhead of the QNAP framework is composed of the addition operations of input activations. Based on EWC, the proposed ECP can further decrease the MACs of CNN models and then improve the overall performance.

First, each 8-b input activation is split into two 4-b data, namely, 4-b I_{HBs} and 4-b I_{LBs}, which are stored separately. In the ECP, I_{HBs} will be all processed (called the execution stage), and the results are used to decide whether the following computations on I_{LBs} (called the prediction stage) are necessary.

The process in ECP to efficiently support the prediction stage is shown in Fig. 6. I_{HBs} are stored in memory bank0. I_{LBs} are alternately stored in memory bank1 and memory bank2, where four $1 \times 1 \times 4$ activations (called AC, from four channels in one row) are combined as the basic storage unit. First, I_{HBs} are processed to implement the execution stage, and the results are compared with the threshold T. If the output value is less than T, it is highly possible that the final result could be negative. Therefore, its flag bit is set as 0, meaning that the corresponding prediction stage can be eliminated. In contrast, if the output is larger than T, the flag bit will be set to 1, namely, the corresponding prediction stage is essential. After one row of results from the execution stage is obtained, the corresponding flag bits are concatenated as the prediction list and stored in registers by taking every two flag bits as a basic unit in each register.

In the prediction stage, eight ACs of I_{LBs} are read out, which should produce six PSums. However, due to the limited

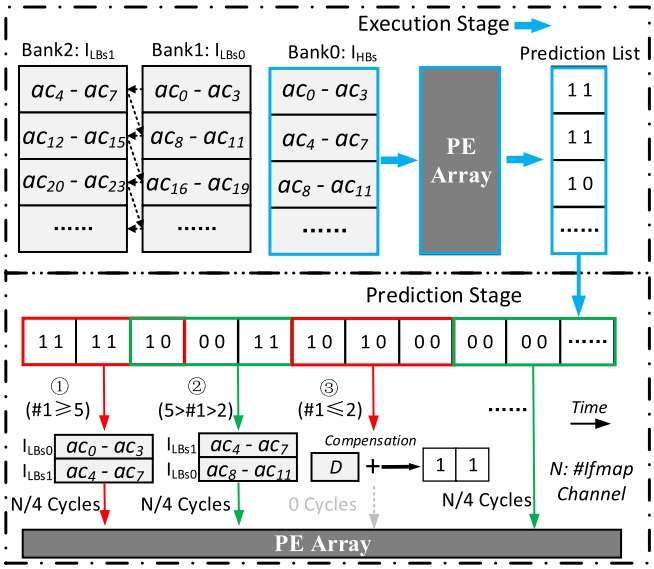


Fig. 6. Process of ECP.

hardware resources, only four PSums can be obtained in each cycle. In fact, some PSums in the six output PSums require no further I_{LBs} computations, namely, their final results are most likely less than zero. To further utilize this property, six successive flag bits in the prediction list are read out in each cycle.

As shown in Fig. 6, there are three cases according to the number of "1" ($\sharp 1$) in the six flag bits:

1) If no less than five "1" are in the six flag bits, the former four I_{HBs} results will be added with I_{LBs} results, and the remaining two flag bits will be combined with the next four flag bits. This step is executed because that most of the six final results are most likely larger than zero. Meanwhile, even if the one (only one "0" in six flag bits) that requires no further I_{LBs}, PSum requires no I_{LBs} computations, and the performance cannot be improved, as only four PSums can be produced by PEA at each cycle. Namely, only five PSums are produced simultaneously, and the performance can be improved. For example, the former I_{HBs} results, $\langle 1111 \rangle$ in $\langle 111110 \rangle$, will be further added with I_{LBs} results, and $\langle 10 \rangle$ will be combined with another four flag bits $\langle 0011 \rangle$, which are processed in the next $N/4$ cycles. N represents the number of Ifmap channels. Note that this step will not degrade accuracy.
2) When the number of "1" in the six flag bits is less than five but larger than two, such as three "1" in $\langle 100011 \rangle$, the index of the AC corresponding to these flag bit "1" values will be obtained. In the 3×3 convolution, as shown in Fig. 6, $\langle 100011 \rangle$ are produced by activations $\langle \mathrm{AC_4AC_5AC_6AC_7AC_8AC_9AC_{10}AC_{11}} \rangle$. The flag bit "1" in $\langle 100011 \rangle$ values corresponds to activations $\langle \mathrm{AC_4AC_5AC_6} \rangle$, $\langle \mathrm{AC_9AC_{10}AC_{11}} \rangle$, and $\langle \mathrm{AC_8AC_9AC_{10}} \rangle$, which are all included in the loaded activations $\langle \mathrm{AC_4AC_5AC_6AC_7AC_8AC_9AC_{10}AC_{11}} \rangle$. Moreover, in this situation, these indexed AC can be all processed at

TABLE II
SPEEDUP AND ACCURACY LOSS ON FOUR BENCHMARKS

Network	**AlexNet**	**VGGNet**	**GoogLeNet**	**ResNet**
Speedup	1.75×	1.54×	1.84×	1.81×
Accuracy Loss	-0.82%	-0.11%	-0.32%	-0.40%

one $N/4$ cycle, achieving more than 75% hardware utilization.

3) If no more than two "1" values are in the six flag bits, such as ⟨101000⟩, the hardware utilization will be lower than 50% if the AC whose flag bits are a value of "1" is processed directly. In this way, the overall performance cannot be improved even though the computations are reduced. To tackle this problem, Song *et al.* [24] proposed a complicated hardware schedule method to improve hardware utilization with a high power consumption overhead. In practice, our experimental results show that, even if all I_{LBs} computations are abandoned in this situation, limited accuracy degradation will occur, as the number of I_{HBs} results that require I_{LBs} computations is small. Meanwhile, a pretrained value D is used in ECP to compensate for the two fewer I_{HBs} results that require I_{LBs} computations to further improve accuracy. Since this situation consumes no cycles in I_{LBs} computations, the performance can be improved.

In other words, ECP achieves more than 75% hardware utilization and then improves the overall performance with acceptable accuracy loss. Meanwhile, as the complicated control logic is eliminated, ECP consumes less power than [24], and the overall performance can be improved with very high hardware utilization due to the lack of time consumption in I_{LBs} computations. Note that the speculation parameter threshold T and compensation value D are first obtained in the inference of CNN models based on the method in [23]. Given the range of parameters (T, D), a gradient descent method is used to find the optimal parameters (T, D) with minimum accuracy loss. Then, the derived parameters (T, D) will be used in the fine-tuning process of CNN models to further improve accuracy. The above process only needs to be executed no more than twice to obtain desirable accuracy.

The ECP can efficiently eliminate potential redundant I_{LBs} computations of activations in the CONV process. Evaluated on four benchmark networks, as shown in Table II, an average of 1.74× speedup can be achieved compared with the direct implementation of I_{HBs} and I_{LBs} computations. It can be learned that most potential redundant I_{LBs} computations can be efficiently eliminated in the proposed ECP with 1% less accuracy loss. Note that the parameters (prediction threshold and compensation value) that determine the speedup and accuracy loss are the optimal results derived from off-line experiments.

V. PIPELINE FOR THE RESIDUAL BLOCK

The residual block is widely used in many mainstream CNN models, which can significantly increase the depth of CNN models and improve the final accuracy. The residual block is mainly formed of three CONV layers, namely, 1 × 1-3 × 3-1 × 1 cascaded CONV layers, as shown in Fig. 7(a). Different from traditional CONV structures, skip connections are built to address the vanishing gradient problem. However, the residual block in hardware implementation requires extra off-chip memory access. This occurs because the Ifmaps have to be read from the off-chip memory twice in the layer-by-layer convolutional mode: one for the first 1 × 1 CONV layer and the other for the addition with the final 1 × 1 CONV layer Ofmaps. Although pipeline modes [6], [16], [20] can effectively eliminate this overhead, low pipeline and hardware utilization are caused in these solutions, which degrades the overall performance.

In this work, the RP mode is proposed to eliminate the above repeated off-chip memory access, as shown in Fig. 7(b). At the start of the residual block, two memories, Ifmap_MEM0 and Ifmap_MEM1, are used to load two-row input activations in the ping-pong mode. Namely, one-row input activations in one memory are read out for the CONV processing of the first 1 × 1, and the other memory is used to load another row activations simultaneously. In methods [6], [16], [20], all CONV layers are processed simultaneously, requiring specific hardware resources assignment for each CONV layer and causing low hardware utilization. In our work, all hardware resources are assigned for only one CONV at one period, which is processed in the same computation dataflow of the layer-by-layer mode. In this way, similar to the layer-by-layer mode, almost 100% hardware and pipeline utilization are achieved, without extra control complexity. First, two-row output activations should be generated in the first 1 × 1 CONV layer and stored in memory Pipeline_0 and Pipeline_1 at period 0. Only in this way can the 3 × 3 CONV layer start with one-row zero-padding input activations. Likewise, at period 1, one-row output activations can be obtained in the 3 × 3 CONV layer and stored in the memory Pipeline_3. These activations are used by the last 1 × 1 CONV layer at period 2. The CONV results will be added to the input activations from the Ifmap_MEM0, eliminating repetitive off-chip memory accesses. Then, sums are written into the output memory. Note that, in each period, only one CONV layer uses all hardware resources to maintain high pipeline and hardware utilization. Here, the processes from period 0 to period 2 are called one iteration.

In the second iteration, only one-row output activations need to be acquired in the first 1 × 1 CONV layer, and they will be stored in Pipeline_2. The activations in Pipeline_0-2 can be combined for the 3 × 3 CONV layer, and the results are then processed in the last 1 × 1 CONV layer. The above process is repeated in the subsequent iterations, where input activations from another row will be written into one of the memories, Pipeline_0-2, at each period 1 in a cyclic mode and activations in the other two memories, will be reused.

The RP mode can significantly reduce not only off-chip memory accesses caused by skip connections across different layers but also the output activation storage of every single layer, such as the structure, 3 × 3 CONV layer-2 × 2 pooling layer-3 × 3 CONV layer. However, this structure requires many more on-chip memories than the residual block in the pipeline mode, as a pooling layer among successive 3 × 3 CONV layers is used. Namely, this overhead may overweigh

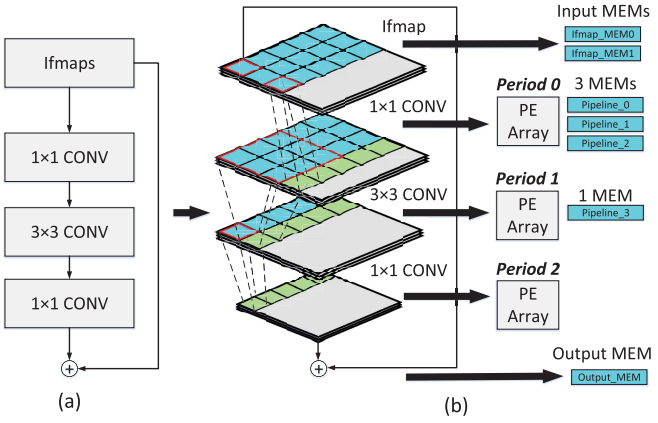


Fig. 7. (a) Architecture of the residual block. (b) RP mode for the residual block.

the benefits from off-chip memory access savings. On the other hand, there are other variant residual blocks with a pooling layer or a stride-2 convolution layer or even a 1×1 convolution in the skip-connection layer. To support these blocks, at least two additional 14-kB SRAMs for the 2×2 pooling layer or the stride-2 1×1 convolution (four 14-kB SRAMs for 3×3 convolution) in the last layer and one 14-kB SRAM (three 14-KB SRAMs for 3×3 convolution), namely, at least 50% (117% with 3×3 convolution), more on-chip SRAM are required. For the inception block in GoogLeNet, 133.3% more on-chip SRAMs are needed. As this extra area consumption overhead will outweigh the benefits brought by the RP mode, the RP mode is more appropriate for the classical block and two/three concatenated layers with $3 \times 3.1 \times 1$ kernels. Since the basic residual block is widely used in mainstream networks, the RP can still bring nonneglectable benefits.

VI. System Architecture

A. Overall Architecture

The overall architecture mainly consists of two components, namely, the memory system and PE array, as shown in Fig. 8. Configuration data are first loaded from off-chip memory during initialization to determine the access pattern in the memory system and computation mode in the PE array. The bitwidth of the activations and weights is chosen as 8 bits to maintain similar accuracy as the original models [30]–[32].

1) Memory System: There are six 14-kB memories for input activations of each CONV layer. Four memories, input feature map (Ifmap) MEM 0-3, are used for the CONV process in layer-by-layer mode. At each cycle, three groups of $1 \times 4 \times 4$ 8-b activations from successive three-row activations in Ifmaps are loaded and stored in Ifmap MEM 0-2. Ifmap MEM 3 is combined with the above three memories as a ping-pong memory group. Namely, the fourth-row activations will be loaded simultaneously into Ifmap MEM 3. After the first three-row activations are processed, the activations in Ifmap MEM 1-3 will be processed, and the fifth row will be loaded into Ifmap MEM 0. The subsequent data loading will follow the above ping-pong mode. In the pipeline mode, six memories are all used. Ifmap MEM 0-1 is used for two-row input activation storage in a residual block. Memory Pipeline_0 (Ifmap MEM 2), Pipeline_1 (Ifmap MEM 3), and Pipeline_2 store three-row input activations for the 3×3 CONV layer. One-row output activations are stored in memory Pipeline_3 for the subsequent 1×1 CONV layer. Specifically, each of the six memories is split into three banks: one 7-kB bank and two 3.5-kB banks. The 7-kB bank is used to store I_{HBs}, and the remaining I_{LBs} are stored in two 3.5-kB banks alternately for the prediction process.

Likewise, two 54-kB weight memories, Weight MEM 0-1, are organized in ping-pong mode to load encoded weights from the off-chip memory and transfer weight_info to the PE array simultaneously. As weight_info in one memory can be highly reused by the PE array, as shown in Fig. 10, sufficient time can be saved for the other memory to load weight_info of the next layer. Therefore, the DRAM bandwidth is set as 128 bits per cycle in QNAP. On the other hand, although weight_info consumes 50% more capacity than that of original weights (21.1% increase of the entire on-chip memory), more than 98% of multiplications and the corresponding power consumption can be saved, which is more appropriate for QNAP design.

Moreover, all results of the layer-by-layer and pipeline modes are written into a 9-kB output memory (output MEM). Once four Ofmaps are finished, the corresponding activations in these four Ofmaps will be written back into off-chip memory. In the convolution process, there are always other components to further process the convolution results, such as the batch normalization (BN) layer and the PReLU function. These components require their own parameters, which are prestored in the 2-kB Param MEM and loaded into the PE array. In addition, as the pooling operations are implemented row by row in our design, a 1-kB Pooling MEM is used to temporarily store one-row pooling results.

2) PE Array: The designed PE array can be configured to support CNN models with different kernel sizes and activation functions, with/without the residual block and BN layer. Meanwhile, two main computation flows, layer-by-layer and pipeline mode, are both supported in the QNAP architecture.

The configurable parameters are first loaded and decoded. The computation-related codes are stored in the configuration buffer and used to generate finite-state machines, which decides and controls the computation mode in the PE array. In the reconfigurable PE array, 144 PEs are further split into 12 pint-sized PE arrays (PEA 0-11), each containing 3×4 PEs. During the convolution process, each of three PEAs is used to process 4-b I_{HBs} or 4-b I_{LBs} of three-row input activations for one output feature map (Ofmap). Therein, I_{HBs} or I_{LBs} of one-row input activations is processed in one PEA, and four PSums can be produced at one cycle. Therefore, PSums of three PEAs will be summed to generate 12-b PSums of four convolution windows.

The four results from three PEAs are further transferred into the component, Weight MUL, to be multiplied with EWs. As 12 PEAs are assigned for 4 Ofmaps at each cycle, four such components, Weight MUL 0-3, are required. When processing I_{HBs} of input activations, the 16-b results of Weight MULs are first stored in 0.5-kB memories (Temp MEM 0-3). After the

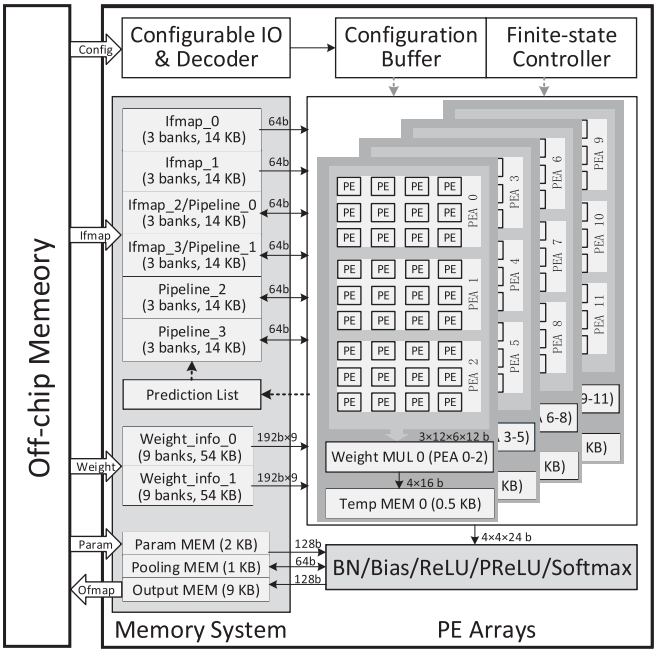


Fig. 8. Overall architecture.

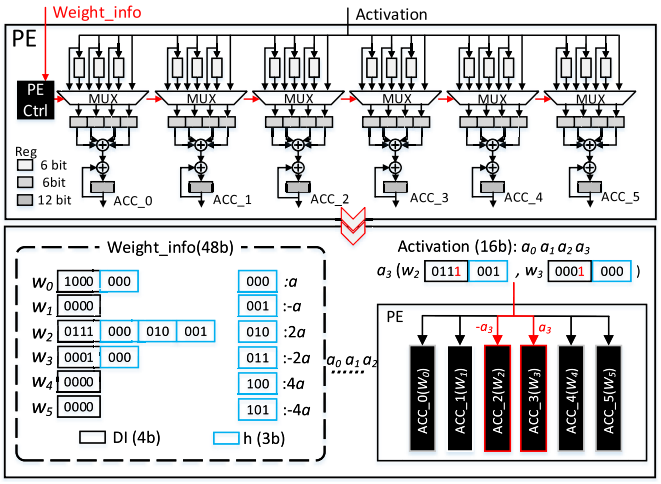


Fig. 9. Microarchitecture of PE.

necessary I_{LBs} of input activations are processed, the PSums will be added with I_{HBs}'s PSums to produce complete convolutional results, which are then further activated (ReLU or PReLU or Softmax) and/or BN and/or pooled (max-pooling) in another component. The final activations will be quantized and written into the output memory.

a) PE microarchitrecture: Four input activations are loaded into one PE at each cycle. Each activation may correspond to one unique weight or different unique weights, and the activation corresponding to the same unique weight should be summed. The first solution is to assign enough accumulators to support the maximum number of unique weights. However, it will severely underutilize hardware resources and area consumption occurs when the number of unique weights is much smaller than the maximum unique weight. Another solution is to use dynamic hardware resource assignment for different numbers of unique weights based on complicated weight encoding. As the dynamic assignment method requires a very complicated logic of resource assignment and data communication, it will cause intolerable power consumption overhead.

Based on the proposed EWC, a novel PE microarchitecture is proposed to address the above problem, as shown in Fig. 9(a). Since four activations are loaded into PE at each cycle and they may be assigned to the same EWs, a four-input processing unit, defined as ACC, is used. In total, six $\mathrm{ACC}_{0,\ldots,5}$ are set for 6 EWs. At cycle t, four preprocessed (negated, left shift, or unaltered) activations are assigned to the corresponding ACC based on the encoded weight_info, as shown in Fig. 4. If fewer than four activations are obtained in each ACC, these activations are first stored in the upper three registers (6 bit), and the adder will not work. At cycle $t+1$, if the number of new input activations and stored activations is not less than four, four of these activations are transferred to the adder for the addition operation. The remaining activations will be stored in the upper registers. The sum will be further accumulated with the previous sum in the 12-b register of ACC. Note that the zero-value input data will be avoided to further reduce unnecessary addition operations.

Combined with encoded weight_info, EWC can be efficiently implemented in the proposed PE microarchitecture. As described in Fig. 4(b), a_3 should be negated for w_2 and unchanged for w_3 based on the weight_info to obtain the same results. Therefore, as shown in Fig. 9(b), a_3 is first preprocessed, and the results ($-a_3$ and a_3) are transferred into accumulators ACC_2 and ACC_3. Note that, if the three registers (6 bit) above the multiplexer of ACC_2 (for example) are all occupied by one activation, a_3 will be summed with these three activations in ACC_2, and the corresponding registers will be released. Otherwise, a_3 will be stored in an idle register. In this way, no accuracy loss will occur in the hardware implementation of the QNAP architecture.

b) Weight MUL module: In QNAP, each PEA generates four 1×3 convolution PSums from the same Ifmap row. Therefore, three PEAs can be combined to form the results of four 3×3 CONV windows. In our work, activations corresponding to the same EW will be first summed up and then multiplied by the EW. The multiplication operations are implemented in the weight MUL module. The details of the Weight MUL module are described with the dataflow in QNAP in Section III-C to provide a better understanding.

B. Computation Flow

1) Dataflow in the Processor: Fig. 10 shows the computation flow of the 3×3 convolution in the proposed processor, which belongs to the weight stationary dataflow. K is the kernel size ($K=3$ here), and W and H represent the width and height of the feature maps, respectively. N and O denote the channel Ifmaps and Ofmaps number, respectively. At each cycle, $3\times4\times4$ activations are processed by 12 PEAs. Here, "3" means the three successive row activations of Ifmap. The first "4" represents the index of activations in the same row; the second "4" means the channel index of activations in different Ifmaps. Each PEA calculates $1\times4\times4$ activations, and

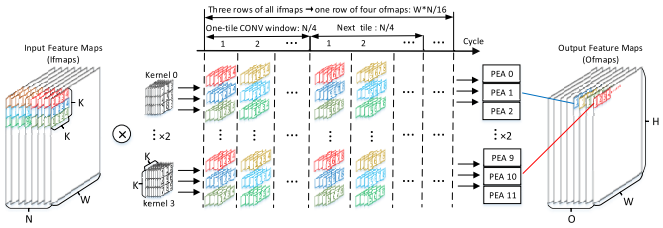


Fig. 10. Computation flow of the 3×3 convolution process.

every three PEAs are combined for one Ofmap. Therefore, all 12 PEAs could be used simultaneously with four Ifmaps and four Ofmaps processed each cycle. For ease of description, $3 \times 4 \times N$ activations are defined as a tile. Only when one tile is completely processed, the next neighbor tile can start. Namely, activations and weights are read along the channel direction in one tile, and $N/4$ cycles are required to consume one tile, as shown in Fig. 10. Therefore, $W * N/16$ cycles are consumed for all tiles in the same row. Then, the abovementioned process is repeated to calculate the following tiles in the next row. Only when the current four Ofmaps are obtained, the next "four" Ofmaps can start to be produced.

2) Dataflow in Each PE Array: In this section, a 3×3 convolution process is used as an example. At each cycle, four $1 \times 1 \times 4$ activations (called AC, from four channels in one row) and three groups of weight_info (called wc, 48 bit, corresponding to original $1 \times 1 \times 4$ weights) are loaded to implement 1×3 convolution in one PEA, which consists of 3×4 PEs. The detailed CONV dataflow for the conventional CONV process (including I_{HBs} computations) in one PEA is shown in Fig. 11.

Similar to Eyeriss [11], weight_info is broadcast across PEs horizontally, and input activations are broadcast across PEs diagonally. In contrast, activations and weight_info are fully reused in QNAP to decrease on-chip memory access. Namely, all activations will be read only once to form one row of Ofmaps in QNAP, while Eyeriss cannot accomplish this. For example, at cycle t, two partial sums (PSums) are directly produced (purple and blue rectangles in Fig. 11). In Eyeriss, AC_3 will be read again to form PSum $\langle \mathrm{AC}_3\mathrm{wc}_0 + \mathrm{AC}_4\mathrm{wc}_1 + \mathrm{AC}_5\mathrm{wc}_2 \rangle$. In contrast, AC_3 are reused to produce incomplete PSums $\langle \mathrm{AC}_2\mathrm{wc}_0 + \mathrm{AC}_3\mathrm{wc}_1 \rangle$ and $\langle \mathrm{AC}_3\mathrm{wc}_0 \rangle$ at cycle t (red rectangles at cycle t) in QNAP. These results are stored and added with the other incomplete PSums at cycle $t+1$ (yellow and green rectangles). In addition to the first cycle, four PSums can be generated in one PEA regularly in sequential cycles with high data reusability. For the 1×1 convolution, the dataflow is similar to the 3×3 dataflow. In contrast, based on the same four input activations from one Ifmap, three PE rows in one PEA can simultaneously generate four results for three Ofmaps. As the results of different PEs do not require a sum operation, they are directly multiplied with weights in the weight MUL module.

In ECP, the dataflow in Fig. 11 can be directly adopted in the execution stage for I_{HBs} computations. However, data communication exists in successive cycles, namely, current PSums (red rectangles in Fig. 11) should be stored until the next cycle. Once the results produced by these PSums

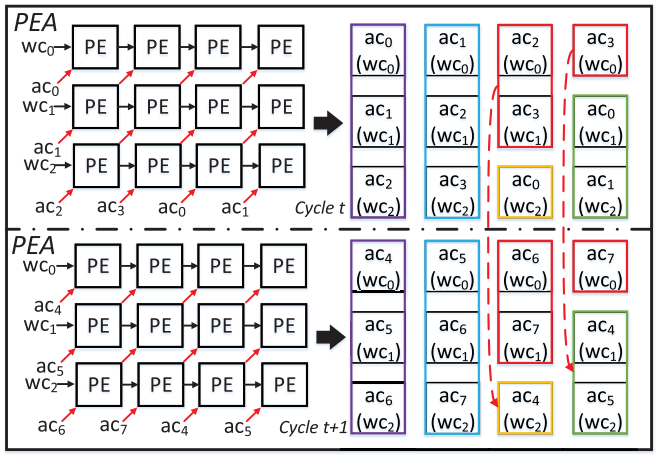


Fig. 11. Dataflow in each PE array for processing the 1×3 convolution.

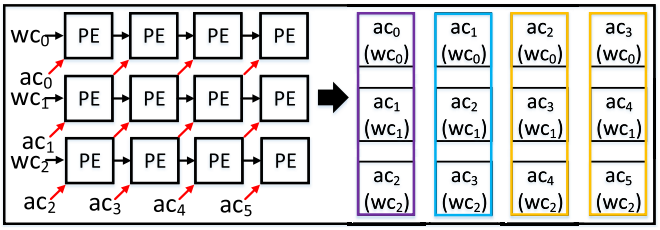


Fig. 12. Dataflow in the prediction stage for processing the 1×3 convolution.

require the prediction stage, two extra cycles are needed, which significantly degrades the final performance. To tackle this problem, dedicated dataflow is proposed, as shown in Fig. 12. Compared with the dataflow in Fig. 11, more activations are used to produce four complete results, which can block the data communication in successive cycles and create favorable conditions for the prediction stage.

To support the above dataflow, the weight MUL module is proposed, and the corresponding microarchitecture is shown in Fig. 13. There are four groups of multiplication-addition-fused (MAF) calculators in the weight MUL module, which correspond to four 3×3 CONV windows. As no more than six EWs remain in our work, each MAF contains six processing units (PU0-5), each dealing with the multiplication operations of one EW. In each PU, all PSums related to the same EW are first added and then multiplied by the EW. The results from all PUs are further added to obtain the final convolutional results. Referring to the dataflow in Fig. 11, the activations in purple and blue rectangles can be directly summed up in one cycle. Therefore, these activations from three PEAs are transferred to the first and second MAFs (MAF0-1). The activations in the yellow rectangle should be added with another two activations (red rectangle) stored at the previous cycle. Therefore, the two activations are first added and stored in the MAF2 and added with activations in the yellow rectangle at the next cycle. Likewise, the activations in the green rectangle should be added with another activation (red rectangle) in the previous cycle. Therefore, the activation in the red rectangle is first stored in MAF3 and added with activations in the green rectangle in the next cycle. For the prediction stage, MAF2-3 are set the same as MAF0-1, which can directly process the activations from three PEAs.

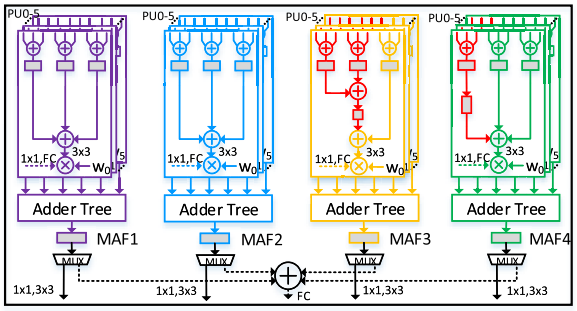


Fig. 13. Microarchitecture of weight MUL module.

When implementing the 1×1 CONV or FC layer, the results from the PEAs are directly multiplied by EWs in MAFs without addition operations. Then, results from each MAF are added to adder trees. In the 1×1 CONV layer, and the sums are directly output. In the FC layer, the sums are further added to generate the final output.

C. Configuration Information

The finite-state controller is used to configure QNAP at three levels: the model level, the layer level, and the PE level, as shown in Fig. 14. First, network information is a 32-b word, where 18 bits are used to indicate the CONV mode (layer-by-layer or pipeline mode), the number of network layers, and the base address of the first layer. Second, the layer information is used to decide the processing of one specific layer. The information contains layer types, kernel size, height, and width of Ifmaps and Ofmaps. In addition, the addresses of weights, Ifmaps, Ofmaps, and so on are also included in the configuration information of this level, which is presented in Fig. 14. Third, PE information directly controls the state of PE to support different CONV modes.

During inference implementation, the finite-state controller first reads the network and layer configuration information in the 320-b configuration buffer. Then, the controller identifies each PEs function by decoding these parameters and chooses the corresponding configuration words for each PE via the point-to-point connections between PE arrays and the configuration buffer.

VII. Chip Implementation and Evaluation

In this section, the proposed techniques and fabricated processors are evaluated. First, the evaluation methodology is described in Section VIII-A. Second, the physical information and overall performance on benchmark networks of QNAP are listed in Section VIII-B. Moreover, the comparisons among the proposed techniques and their related methods, and the overall comparison results between QNAP and other SOTA processors are shown in Section VIII-C.

A. Evaluation Methodology

To prove the effectiveness and generality of the QNAP, four benchmark CNN models, AlexNet [22], VGGNet [4], GoogLeNet [19], and ResNet-50 (called ResNet here) [18], are taken as the benchmarks. With the ImageNet [33] dataset used,

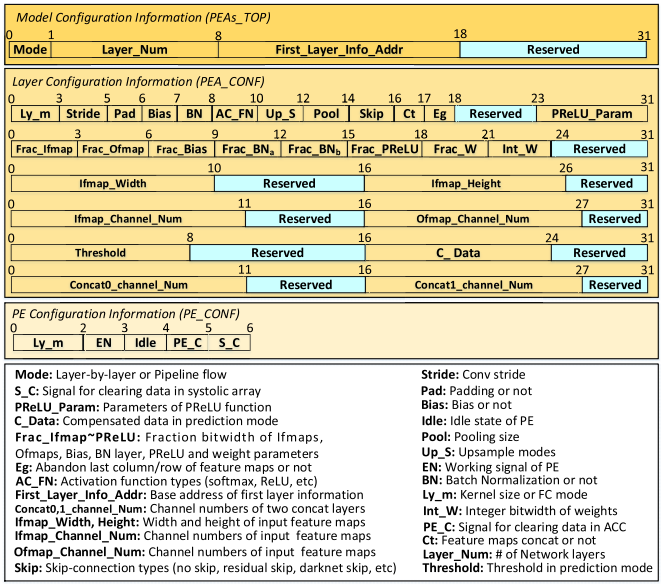


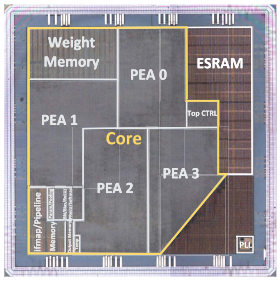
Fig. 14. Three-level configuration hierarchy: model information, layer information, and PE information.

these models are trained and quantized in TensorFlow [34]. Considering all related works [17], [23], [24] take the Eyeriss architecture as the baseline, with only relative results provided, an Eyeriss-like architecture is also built in this work to fairly compare each contribution of QNAP with the corresponding related works. Note that, because EWC and ECP are proposed to reduce the number of multiplication and addition operations separately, each multiplication or addition is counted as one operation in this work to explicitly evaluate the advantages of the proposed methods. Consequently, all the operation-related metrics given in referenced processors are all transformed under this definition for a fair comparison. For example, the energy efficiency value of compared processors where one MAC only represents one operation will be doubled. Meanwhile, the number of operations is counted based on the actual number executed in QNAP for a fair comparison with other processors, as some processors also use the actual number of operations instead of the original operation number required by the original models.

The overall performance and power of the QNAP are obtained through chip evaluation. To evaluate the effect of each technique in QNAP, dedicated architectures are designed to fairly compare with their related works. The corresponding area and performance are obtained based on the Synopsys Design Compiler with a TSMC 28-nm standard-cell library used. The power consumption is obtained based on the post-layout-based simulation. Referring to [23], the publicly available Micron's DDR4 system power calculator [35] is used to estimate the power cost of accesses to the off-chip memory.

B. Chip Implementation

QNAP is fabricated in 28-nm CMOS technology, and the die photograph and chip specification are shown Fig. 15. The performances of QNAP evaluated on the four benchmark networks are listed in Table III. In fact, the bitwidth of weights mainly decides the number of unique weights. As no more



Technology	TSMC 28nm
Chip Size (mm^2)	1.8×1.8
Core Size (mm^2)	1.9
Logic Gates (MGE)	3.7
SRAM (KB)	206
Voltage (V)	0.6 - 0.9
Frequency (MHz)	100 - 470
Power (mW@V,MHz)	131.6 @ 0.9, 470 19.4 @ 0.6, 100
Data Width (b)	8
EnergyEfficiency (TOPS/W@V,MHz)	12.1 @ 0.9, 470 17.5 @ 0.6, 100

Fig. 15. Layout and physical information of QNAP.

TABLE III
PERFORMANCE ON FOUR BENCHMARK NETWORKS AT 470 MHz AND 0.9 V

Networks	AlexNet	VGGNet	GoogLeNet	ResNet
Bit-width	8	8	8	8
Accuracy*	53.60%	70.94%	68.21%	76.92%
Throughput (FPS)†	306.3	4.3	118.4	40.4
Power (mW)	140.4	136.0	128.7	125.8
Energy Efficiency (TOPS/W)	11.30	11.67	12.33	12.62

* Top-1 accuracy on ImageNet.
† Frames per second.

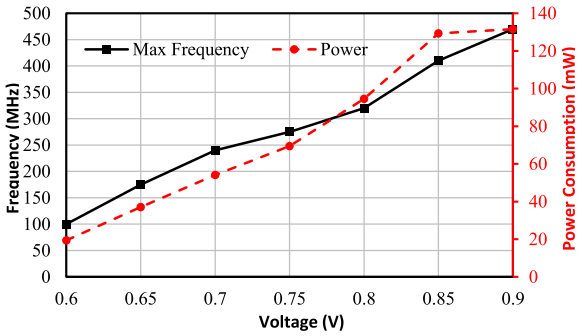


Fig. 16. Max frequency and power consumption when voltage scales.

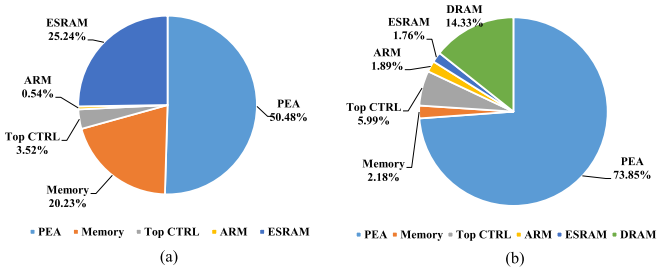


Fig. 17. (a) Area breakdown and (b) power breakdown of the processor.

than six EWs are further selected from unique weights in ECP, weights with different bitwidths achieve similar performance in QNAP. Therefore, although QNAP can support quantized weights from 1 to 8 bits, here, 8 bits, the most widely used bitwidth [36], [37] is used for evaluation to obtain high accuracy without loss of generality.

Similar power consumption occurs in all networks. In contrast, GoogLeNet and ResNet consume relatively less power than VGGNet and AlexNet because the former models adopt much more 1 × 1 CONV layers, which achieves higher reusability of input activations than 3 × 3 CONV layers. Moreover, the proposed RP mode can significantly reduce off-chip memory access required by the skip-connection in the residual and inception blocks, decreasing the corresponding costly power consumption. The throughput of VGGNet is only 4.3 FPS (frame per second) due to the highest computing complexity among the four models (more than 23× than AlexNet).

1) Analysis on Voltage Scaling: The power consumption and maximum frequency of this chip under different supply voltages are tested in this section. The power consumption is the average results of AlexNet, VGGNet, GoogLeNet, and ResNet. Fig. 16 shows the voltage-max frequency and power consumption scaling results. When the voltage is scaled up from 0.6 to 0.9 V, the maximum frequency of the processor ranges from 100 to 470 MHz. Meanwhile, the corresponding power consumption varies from 19.4 to 131.6 mW. The power of the QNAP grows proportionally with the work voltage and frequency.

2) Analysis of Area and Power Breakdowns of the Processor: The area and power breakdowns of the processor with ResNet are evaluated, which are obtained by postlayout simulations at 470 MHz, as illustrated in Fig. 17. The 12 PEAs, namely, 144 PEs, take up 50.48% of the area and 86.21% power consumption since PEAs consume a number of registers and most compute units to support most operations. The memory consumes an area of 20.3% and 2.55% of the power to store and transmit the weights, activations, and intermediate PSums. The low power consumption of memory is attributed to the high data reusability of weight and input activations in the proposed dataflow. The other parts show small area and power consumptions, including ESRAM for verification and the ARM core occupation. Due to the high reusability of parameters and input activation in convolutions, most off-chip DRAM power consumption comes from parameter-intensive fully connected layers. Because fully connected layers consume only a negligible part of the overall computations, the whole off-chip DRAM power consumption is relatively low (approximately 14.33%). Note that both the following power consumption and area refer to that of the core chip, excluding the ARM, ESRAM, and DRAM (the same as prior works [31], [38], [39]).

3) Analysis on Area and Power Breakdowns of PE: The area and power breakdowns of a PE running inference of ResNet are shown in Fig. 18. The PE mainly consists of the PE controller, adders, registers, and multiplexer, which ensures the flexibility and computational efficiency of the PE. PE control accounts for the largest proportion of the area consumption, nearly 44.06%, as the control unit is used to assign the input activations to the specific registers according to the corresponding decoded weight information. The control signal is distributed all over the PE. In contrast, 19.30% of the power is consumed by PE control, while registers consume approximately 73.22% of the total power. The reason is that the input activations are stored in the registers at each cycle; therefore, only when the four activations are ready, the corresponding accumulator can work. The PSums are also first stored in the registers and accumulated with another group of PSums until all PSums corresponding to the same Ofmap are summed. The process above requires an amount of register

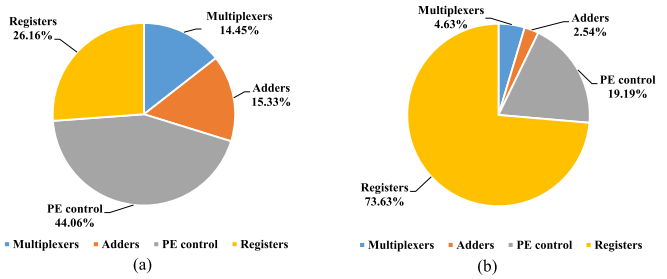


Fig. 18. (a) Area breakdown and (b) power breakdown of the PE.

access, causing a higher power consumption proportion of registers in the processor.

C. Comparison With State-of-the-Art Works

1) Weight Elimination-Based Methods: In this section, the proposed EWC is compared with the most related work, UCNN [17], which is the first work that explores the repetition of quantized weights. Note that the relative results compared with Eyeriss are provided in UCNN, and relative comparison results of EWC to Eyeriss are also presented. Meanwhile, EWC adopts a standard quantization technique without a retraining process, the same as the UCNN_{256}. Our result is obtained by the post-layout simulation after placing-and-routing with the only EWC adopted for a fair comparison.

The comparison results are shown in Table IV. With the TTQ quantization technique [40] and retraining process adopted, both of which can significantly reduce the number of unique weights, UCNN_3 performs best compared with its other versions in energy efficiency. However, TTQ quantization loses several percent in accuracy on ResNet, which is unacceptable. In contrast, though standard quantization is adopted in EMC without a retraining process, energy efficiency can be improved by 1.59× on AlexNet and 1.78× on ResNet without accuracy loss compared to UCNN_3. The energy efficiency of EWC is also better than that of other versions of UCNN. Although INQ quantization technique [41] and retraining process are used in UCNN_{17}, EWC performs better on AlexNet and ResNet (2.05× on average). In particular, when the same quantization technique is used, EWC achieves approximately 3.15× higher energy efficiency than UCNN_{256} (not retrained).

There are three primary reasons that the energy efficiency of EWC is higher than that of UCNN [17]. First, UCNN focuses on the elimination of repetitive computations caused by the same weights of different kernels in one Ifmap, which can only explore a small portion of potential repetitive multiplications. In contrast, repetitive quantized weights in one kernel are fully exploited in EWC, causing the result that on average, multiplications are eliminated by 2.4× in UCNN compared with 29.1× in EWC. Second, different Ifmaps require different group lists in UCNN, which are produced online with additional control complexity and latency. In contrast, EW selection and encoding operations in EWC are all finished offline. Third, hardware resources are assigned according to the maximum number of quantized weights in the UCNN, causing significant resource waste. Due to the weight decomposition technique in EWC, no more than six EWs remain in the end. On the other hand, without a group list and the corresponding intermediate data in UCNN, EWC only requires 39.8% of on-chip memory and an area of 52.8% compared with UCNN, demonstrating better area efficiency of EWC.

TABLE IV
ENERGY EFFICIENCY COMPARISON WITH UCNN

Energy Efficiency	AlexNet	VGGNet	GoogLeNet	ResNet
UCNN_3* [17]	1.75×	-	-	1.90×
UCNN_{17}* [17]	1.34×	-	-	1.67×
UCNN_{64}* [17]	0.96×	-	-	1.14×
UCNN_{256}* [17]	0.87×	-	-	1.09×
EWC	2.78×	3.35×	3.33×	3.39×

	♯PE	♯MUL	Memory	Area
UCNN_{32nm} [17]	32	256	512 KB	3.6 mm^2
EWC_{28nm}	144	96	204 KB	1.9 mm^2

* xx in UCNN_{xx} indicates various weight group numbers corresponding to different quantization techniques. 3: TTQ; 17: INQ; 64: standard quantization techniques with the retraining process; 256: standard quantization techniques without the retraining process.

TABLE V
ENERGY EFFICIENCY COMPARISON WITH SNAPEA AND PRED

Energy Efficiency	AlexNet	VGGNet	GoogLeNet	ACC Loss
SnaPEA_{28nm} [23]	1.56×	1.32×	1.62×	3%
Pred_{FPGA} [24]	1.1×	1.4×	-	0
ECP	1.92×	1.70×	1.99×	<1%

TABLE VI
COMPARISON WITH OTHER PIPELINE-BASED METHODS AT 470 MHz, 0.9 V, AND 28 nm

	Memory	Hardware Utilization	Power
Fused [16]	140 KB	84.6%	157.9 mW
Multi-Face [6]	112 KB	89.1%	149.1 mW
RP	86 KB	100%	131.6 mW

2) Prediction-Based Methods: In this section, the two most related works, SnaPEA [23] and Pred [24], are compared with the proposed ECP method, and the results are listed in Table V. Compared with SnaPEA [23], an average of 1.25× energy efficiency improvement is achieved on AlexNet, VGGNet, and GoogLeNet by the ECP. Meanwhile, less than a 1% accuracy loss is obtained, while SnaPEA loses approximately 3% accuracy to achieve comparable energy efficiency. The reason is that the computational imbalance between weights and activations in SnaPEA is still not eliminated completely, prohibiting further performance and accuracy improvements. Although there is no accuracy loss, Pred [24] requires a complicated pooling layer and an ReLU function-based lookup table to alleviate workload imbalance, causing high power and latency. Therefore, its energy efficiencies on AlexNet and VGGNet are 1.75× and 1.22× lower than that of ECP. Furthermore, as there is no pooling layer in the residual and inception blocks, Pred [24] cannot be used for these blocks in ResNet and GoogLeNet. In contrast, the proposed ECP has no such limitations and can be generally applied for current mainstream networks.

3) Pipeline-Based Methods: In this section, the two most related works, Fused [16] and Multi-Face [6], are compared with the proposed RP mode. The three methods are achieved and applied for a canonical residual block based on the EWC

TABLE VII
COMPARISON WITH OTHER SOTA PROCESSORS[♮]

	VLSI'18 [31] (Sticker)	**ISSCC'18** [30] (UNPU)	**VLSI'19** [42] (SNAP)	**ISSCC'19** [43] (Butterfly)	**ISSCC'19** [44] (DRL)	**ISSCC'20** [32] (Dual-core)	**QNAP**
Function	CONV+FC	CONV+FC	CONV+FC	CONV+FC	CONV+FC	CONV+FC	CONV+FC
Process (nm)	65	65	16	8	65	7	28
Area (mm^2)	7.8	16	2.4	5.5	16	3.04	1.9
Supply Voltage (V)	0.67 - 1	0.63 - 1.1	0.55 - 0.80	0.5 - 0.8	0.67 - 1.1	0.575 - 0.825	0.6 - 0.9
Frequency (MHz)	20-200	2 - 200	33 - 480	67 - 933	5 - 200	290 - 880	100 - 470
Memory (KB)	170	256	280.6	1568	448	2176	206
Bit Precision	8	1 - 16	16	8, 16	16	8	1 - 8
Power (mW)	20.5 - 248.4	3.2@5M,0.635V 297@200M,1.1V	16.3 - 364	39 @ 0.5V, 1533 @ 0.8V	196@200M,1.1V 2.4@10M,0.67V	174 - 1053 @ 0.575V - 0.825V	19.4 - 140.4 @ 8b
Energy Efficiency† (TOPS/W)	0.5 @ 1V, 200MHz	4.66 @ 8b, 1.1V, 200MHz	1.67 @ 0.55V, 260MHz	2.46 @ 8b, 0.8V	4.32	6.84-13.66 @ 0.825V-0.575V	12.62 @ 8b, 0.9V, 470MHz
Energy Efficiency-N† (TOPS/W)	0.62 @ 0.9V, 200MHz	6.96 @ 8b, 0.9V, 200MHz	0.62 @ 0.9V, 260MHz	1.94 @ 8b, 0.9V	6.5	5.58 @ 0.9V	12.62 @ 8b, 0.9V, 470MHz
Frame Energy‡ (mJ/Frame)	9.69	41.32 @ 8b, 1.1V, 200MHz	6.02 @ 0.8V, 480MHz	3.21 @ 8b, 0.8V, 933MHz	1.91 @ 8b, 1.1V, 200MHz	2.34 @ 8b, 0.825V, 880MHz	0.46 @ 8b, 0.9V, 470MHz
Frame Energy-N‡ (mJ/Frame)	7.85	27.66 @ 8b, 0.9V, 200MHz	7.62 @ 0.9V, 480MHz	4.06 @ 8b, 0.9V, 933MHz	1.28 @ 8b, 0.9V, 200MHz	2.78 @ 8b, 0.9V, 880MHz	0.46 @ 8b, 0.9V, 470MHz
Area Efficiency[♭] ($GOPS/mm^2$)	14.3	77.42 @ 8b	118.6	622.9 @ 8b	25.5	2125.6 @ 8b	745.1 @ 8b
Area Efficiency-N[♭] ($GOPS/mm^2$)	65.2	353.1 @ 8b	48.8	172.9 @ 8b	116.3	373.6 @ 8b	745.1 @ 8b

[♮] Dense and non-pruned networks are used for evaluation; -N: normalized to 28 nm through the general scaling.
‡ The performance (FPS) of other processors is estimated assuming their hardware utilization is 100%.
[♭] Normalized area ratio (7nm:8nm:16nm:28nm:40nm:65nm) is assumed to be (1:1.58:2.34:5.69:8.57:25.95) [45], [46]. Normalized to 28 nm, 470 MHz.
† [♭]One addition and one multiplication represent two ops.

and ECP, and the results are listed in Table VI. Compared with Fused [16] and Multi-Face [6], the RP mode can not only improve hardware utilization by 15.4% and 10.1% but also require 38.6% and 23.2% fewer feature map memories. The reason is that the fixed pyramid structure in [16] and [6] causes relatively lower hardware utilization and more memory consumption. Furthermore, dispersive hardware resources for each layer in [16] and [6] lower data reuse, causing more memory access and power consumption, respectively, namely, 16.6% and 11.7% more power consumption than the RP mode.

4) Comparison With State-of-the-Art Processors: Table VII details the comparisons with the SOTA CNN processors. As four dense networks (AlexNet, VGGNet, ResNet, and GoogLeNet) are used as the benchmarks, the best energy efficiency (TOPS/W) and frame energy (mJ/Frame) results among these four models are listed in Table VII to show the superiority of QNAP. The results of QNAP are obtained based on 8-b quantized CNN models. Because EWC eliminates most multiplications in network inference, only 96 multipliers are required to support a small number of multiplications in QNAP. Furthermore, the RP mode can avoid most off-chip memory access in the residual block of ResNet. Therefore, the corresponding power and area consumption can be reduced. QNAP's energy efficiency achieves 12.56 TOPS/W when operating at 470 MHz and 0.9 V with ResNet evaluated. Meanwhile, QNAP outperforms recent processors from 1.85× [32] to 25.24× [31] and 1.65× [32] to 22.4× [31] for AlexNet. Meanwhile, 745.1 $GOPS/mm^2$ area efficiency is achieved. Normalized to 28 nm, QNAP also performs better in the area efficiency, from 1.99× [32] to 15.27× [42]. To further compare the effective performance of all processors, frame energy is used as the third metric. As this merit is not provided in most works, we refer to the number of processing units in each processor and the MAC operations in four networks to obtain processors' performance. Note that the performance of QNAP is obtained based on actual measurements. Assuming that the hardware utilization of all compared processors is 100%, QNAP achieves 0.46 mJ/Frame and performs best with AlexNet evaluated, decreasing frame energy by at least 4.16× [44] and at most 89.77× [30] with AlexNet evaluated on QNAP while at least 0.06× [44] and at most 1.31× [30] with VGGNet evaluated on QNAP. Note that the adopted normalized area ratio refers to paper [45] and report [46].

VIII. CONCLUSION

In this article, an energy-efficient CNN processor, namely, QNAP, is proposed together with a set of algorithmic and hardware optimizations based on a thorough analysis of the bottlenecks in current mainstream CNN models. First, EWC is proposed to eliminate most of the multiplications caused by repetitive quantized weights with no more than 6 EWs. Second, ECP is presented to further reduce addition operations by removing the potential redundant operations caused by the ReLU function. Furthermore, the RP mode is further presented to achieve memory-efficient pipelined processing of residual blocks with nearly full hardware utilization. Finally, to verify the proposed techniques, the corresponding quantized CNN processor is fabricated based on a dedicated architecture design. In future work, the repetition of quantized activations and larger quantization bitwidths will be taken into consideration for further improvement in the energy efficiency of CNN implementations.

REFERENCES

[1] J. Vongkulbhisal, F. De la Torre, and J. P. Costeira, "Discriminative optimization: Theory and applications to computer vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 4, pp. 829–843, Apr. 2019.

[2] S. Liu, Z. Li, V. Srikumar, V. Pascucci, and P.-T. Bremer, "NLIZE: A perturbation-driven visual interrogation tool for analyzing and interpreting natural language inference models," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 1, pp. 651–660, Jan. 2019.

[3] X. Xu, J. Deng, E. Coutinho, C. Wu, L. Zhao, and B. W. Schuller, "Connecting subspace learning and extreme learning machine in speech emotion recognition," *IEEE Trans. Multimedia*, vol. 21, no. 3, pp. 795–808, Mar. 2019.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Comput. Sci.*, vol. 2014, pp. 1–24, Sep. 2014.

[5] A. Ren *et al.*, "ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 925–938.

[6] H. Mo *et al.*, "A multi-task hardwired accelerator for face detection and alignment," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 11, pp. 4284–4298, Nov. 2020.

[7] S. Yin *et al.*, "An energy-efficient reconfigurable processor for binary- and ternary-weight neural networks with flexible data bit width," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 1120–1136, Apr. 2018.

[8] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 99, pp. 1–12, Nov. 2017.

[9] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.

[10] S. Choi, J. Lee, K. Lee, and H.-J. Yoo, "A 9.02 mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 220–222.

[11] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H. J. Yoo, "A 0.62 mW ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on haar-like face detector," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 248–249.

[12] H. T. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 821–834.

[13] A. D. Lascorz *et al.*, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 749–763.

[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[15] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[16] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[17] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "UCNN: Exploiting computational reuse in deep neural networks via weight repetition," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 674–687.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[19] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[20] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[21] H. Mo *et al.*, "9.2 A 28 nm 12.1TOPS/W dual-mode CNN processor using effective-weight-based convolution and error-compensation-based prediction," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 146–148.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[23] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2018, pp. 662–673.

[24] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction based execution on deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 752–763.

[25] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2990–2999.

[26] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2217–2225.

[27] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, "Exploiting cyclic symmetry in convolutional neural networks," 2016, *arXiv:1602.02660*. [Online]. Available: https://arxiv.org/abs/1602.02660

[28] S. Yu, K. Wickstrøm, R. Jenssen, and J. C. Príncipe, "Understanding convolutional neural networks with information theory: An initial exploration," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 1–8, Jan. 2020.

[29] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization," in *Proc. 53rd Annu. Design Automat. Conf.*, Jun. 2016, pp. 1–6.

[30] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 218–220.

[31] Z. Yuan *et al.*, "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," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 33–34.

[32] C.-H. Lin *et al.*, "A 3.4-to-13.3TOPS/W 3.6TOPS dual-core deep-learning accelerator for versatile ai applications in 7 nm 5G smartphone SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 134–136.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[34] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[35] *DDR4 Spec—Micron Technology*. Accessed: Aug. 2016. [Online]. Available: https://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/

[36] J. Albericio *et al.*, "Bit-pragmatic deep neural network computing," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 382–394.

[37] S. Migacz, "NVIDIA 8-bit inference width TensorRT," in *Proc. GPU Technol. Conf.*, vol. 2, no. 4, 2017, p. 5.

[38] K. Ueyoshi *et al.*, "QUEST: A 7.49 TOPS multi-purpose log-quantized DNN inference engine stacked on 96 MB 3D SRAM using inductive-coupling technology in 40 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2018, pp. 216–218.

[39] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67—21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C306–C307.

[40] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: https://arxiv.org/abs/1612.01064

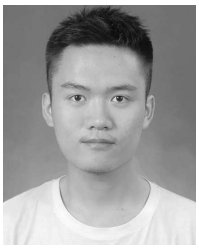
[41] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*. [Online]. Available: https://arxiv.org/abs/1702.03044

[42] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67-21.55TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C306–C307.

[43] J. Song *et al.*, "An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8 nm flagship mobile SoC," *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 130–132.

[44] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1TOPS/W mobile deep RL accelerator with transposable PE array and experience compression," *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 136–138.

[45] Y. Yamada *et al.*, "7.2 A 20.5TOPS and 217.3GOPS/mm^2 multicore SoC with DNN accelerator and image signal processor complying with ISO26262 for automotive applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 132–134.
[46] TSMC. (2019). *TSMC 2019 Annual Report*. [Online]. Available: https://investor.tsmc.com/static/annualReports/2019/english/ebook/index.html



Huiyu Mo received the B.S. degree from the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently pursuing the Ph.D. degree with the Institute of Microelectronics, Tsinghua University, Beijing, China.

His current research interests include reconfigurable computing, mobile computing, and VLSI signal processing and computer vision.

Wenping Zhu (Member, IEEE) received the B.S. degree from the School of Microelectronics, Xidian University, Xi'an, China, in 2008, and the Ph.D. degree from the Institute of Microelectronics, Tsinghua University, Beijing, China, in 2016.

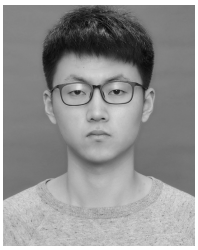
He is currently an Assistant Professor with the Institute of Semiconductors, Chinese Academy of Sciences, Beijing. His main research interests include mobile computing and VLSI SoC design, multi-sensor fusion, and hardware-oriented mobile vision algorithm optimization and design.

Wenjing Hu received the B.S. degree from the Department of Electrical Engineering, Yanshan University, Qinhuangdao, China, in 2012, and the M.S. degree from the Institute of Microelectronics, Tsinghua University, Beijing, China, in 2015.

She is currently a Senior ASIC Design Engineer with Beijing AI Chip Technology Ltd. Company, Beijing. Her current research interests include the design and implementation of deep learning accelerator processors and reconfigurable computing.

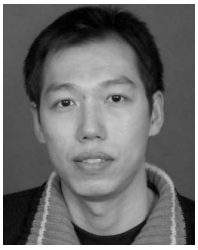
Qiang Li received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1999 and 2002, respectively.

He is currently working on sports-related computer vision projects in Intel Corporation, Beijing. His current research interests include computer vision, graphics, and hardware-oriented software optimization.



Ang Li received the B.S. degree from the School of Microelectronics, Xidian University, Xi'an, China, in 2019. He is currently pursuing the M.S. degree with the Institute of Microelectronics, Tsinghua University, Beijing, China.

His research interests include computer vision, deep learning, deep neural network (DNN) accelerator, and reconfigurable computing.



Shouyi Yin (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2000, 2002, and 2005, respectively.

He was a Research Associate with Imperial College London, London, U.K. He is currently an Associate Professor with the Institute of Microelectronics, Tsinghua University. His research interests include mobile computing, wireless communications, and SoC design.

Shaojun Wei (Fellow, IEEE) was born in Beijing, China, in 1958. He received the Ph.D. degree from the Faculte Polytechnique de Mons, Mons, Belgium, in 1991.

He became a Professor at the Institute of Microelectronics, Tsinghua University, Beijing, China, in 1995. His main research interests include VLSI SoC design, electronic design automation (EDA) methodology, and communication ASIC design.

Dr. Wei is also a Senior Member of Chinese Institute of Electronics (CIE).

Leibo Liu (Senior Member, IEEE) received the B.S. degree in electronic engineering and the Ph.D. degree from the Institute of Microelectronics, Tsinghua University, Beijing, China, in 1999 and 2004, respectively.

He is currently a Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and very-large-scale integration digital signal processing.