DRQ: Dynamic Region-based Quantization for Deep Neural Network Acceleration

Zhuoran Song[1], Bangqi Fu[1], Feiyang Wu[1], Zhaoming Jiang[1], Li Jiang[1], Naifeng Jing[1]*, Xiaoyao Liang[1,2]
[1] *Shanghai Jiao Tong University, China,* [2] *Biren Research, China*

***Abstract*—Quantization is an effective technique for Deep Neural Network (DNN) inference acceleration. However, conventional quantization techniques are either applied at network or layer level that may fail to exploit fine-grained quantization for further speedup, or only applied on kernel weights without paying attention to the feature map dynamics that may lead to lower NN accuracy. In this paper, we propose a dynamic region-based quantization, namely DRQ, which can change the precision of a DNN model dynamically based on the sensitive regions in the feature map to achieve greater acceleration while reserving better NN accuracy. We propose an algorithm to identify the sensitive regions and an architecture that utilizes a variable-speed mixed-precision convolution array to enable the algorithm with better performance and energy efficiency. Our experiments on a wide variety of networks show that compared to a coarse-grained quantization accelerator like "Eyeriss", DRQ can achieve 92% performance gain and 72% energy reduction with less then 1% accuracy loss. Compared to the state-of-the-art mixed-precision quantization accelerator "OLAccel", DRQ can also achieve 21% performance gain and 33% energy reduction with 3% prediction accuracy improvement which is quite impressive for inference.**

I. Introduction

Facing the increasing demand on the computing power for DNNs, quantization has been acknowledged as an effective technique for reducing the inference workload, and spawns a number of accelerator designs, such as Eyeriss [5], BitFusion [29] and OLAccel [26]. Quantization techniques, by means of transforming floating point into integer data or clustering data into groups [15], [17], [22], [37], [38], can greatly reduce the workload and the required memory bandwidth, and therefore have an overall benefit on performance, energy and the silicon area for the accelerators.

As illustrated in Fig. 1, the conventional quantization techniques [15], [17], [26], [33], [38], as well as OLAccel [26] try to quantize the NN models based on the static distribution of the weight values, and are succeed in reducing the model precision from FP32 to INT8, or even to INT2. Since the weight values can be trained beforehand, this simplifies the quantization scheme and makes the solution fixed at the beginning of any kernel execution. On the other hand, the input feature map values are also important and have varying degrees of impact on NN as reported in works [10] and [3]. However, the existing quantization techniques fail to dynamically capture the sensitivity and variability in the input feature map values, so that these works end up missing the chance in further reducing the power and improving accuracy. Therefore, we are motivated to apply different levels of quantization for different sensitivity in the feature map values.

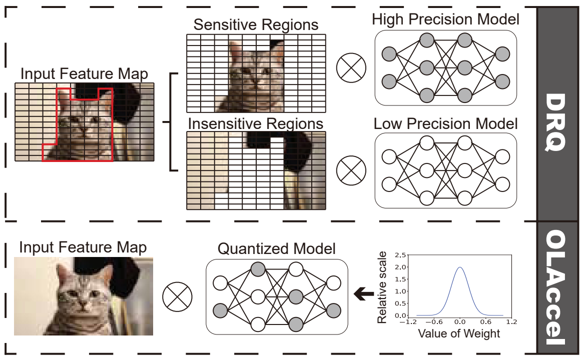


Fig. 1: Comparison of DRQ and other quantization schemes.

In this paper, we propose a novel dynamic region-based quantization, termed as DRQ, to capture the feature characteristic in the input feature map for quantization. As shown in Fig. 1, the dynamic quantization can dynamically track the sensitive regions in the input feature map which are likely to contain important feature information for the final inference results, and smartly adjust the weight and activation precision for computation to different levels according to the sensitivity. In this way, the characteristics of the input feature map will be reserved and fortified for better NN accuracy and meanwhile saving significant amount of computing resources and power from the non-sensitive regions.

Our study mainly answers the following three questions to make the proposed quantization method practical and efficient:

- Is it possible to conduct an input region-based quantization for better NN accuracy? The answer is yes. Based on our study in Section II, we find that some of the input values are highly correlated with the NN accuracy, termed as the sensitive values. In addition, we find that these sensitive values tend to aggregate in a region. The observation implies that applying high-fidelity quantization on the sensitive regions will reserve the NN accuracy, while applying low-fidelity quantization on the insensitive regions can boost the performance.
- How to find the sensitive regions for the dynamic quantization? In Section III, we propose an algorithm to identify the sensitive regions at run time. The algorithm can be instantiated as a just-in-time predictor, which can be neatly embedded after each convolution layer to predict the sensitivity of the input values for the next layer.
- How to conduct an efficient dynamic quantization? In

978-1-7281-4661-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ISCA45697.2020.00086

Method		DRQ	OLAccel [26]	BitFusion [29]	Eyeriss [5]
Dynamic Quantization		√	×	×	×
Quantization Granularity	Network-wise	√	√	√	√
	Layer-wise	√	√	√	×
	Region-wise	√	×	×	×
	Value-wise	√	√	×	×
Bit-width		4/8 bit	4/8 bit	1/2/4/8 bit	16 bit

TABLE I: Comparison of different quantization methods.

Section IV, we propose an accelerator design to support the dynamic region-based quantization, mainly consisting of a variable-speed systolic convolution array, which can switch the execution mode based on the sensitivity identified by the predictor. It applies a simple but fine-grained control scheme that can perform high-precision computation for the sensitive regions and low-precision computation for the insensitive regions.

For a better understanding of our DRQ scheme, we compare it with other quantization techniques as in Table. I. Different from the conventional static quantization [5], [26], [29], DRQ **dynamically** quantizes weights and input values based on the sensitive regions that can better reserve the NN accuracy. The idea is to make the high-precision computation happening to the right place in the feature map. Since the input images vary according to the realtime applications, only dynamic schemes can capture the facts per image basis. Conventional schemes only support network- or layer-wise quantization. But DRQ can support **sub-layer** and even **sub-feature-map** quantization, which can provide more freedom for fine-grained tuning. Finally, DRQ is **hardware friendly** and can be realized by slightly modifying the widely used systolic convolution array, largely reserving the NN accuracy and saving power. In summary, DRQ facilitates an efficient accelerator design with a tightly coupled algorithm and hardware architecture for image classification workloads.

The remaining of this paper is organized as follows. Section II introduces the motivation of our proposed design. Section III describes the sensitivity aware convolution algorithm and Section IV elaborates the proposed microarchitecture. Section V and VI present the experimental methodology and the experiment results. Section VII reviews the related work. Finally, Section VIII concludes this paper.

II. Identifying Sensitivity Region in the Input Feature Maps

According to the value distribution in the input feature maps of the convolution neural networks (CNNs), it has been observed that the majority of values in input feature map after batch normalization and ReLU activation function are close to or exactly zero, while a small number of values are large. Because quantizing values impacts the NN accuracy as revealed by prior studies [26], [27], two questions naturally arise: 1) whether there are certain values in input feature maps that are important to the NN accuracy? And how these values affect the accuracy and to what extend; 2) what are the locations of these values in the input feature maps and how are they spatially distributed?

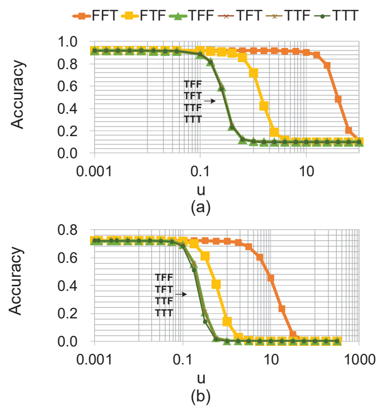


Fig. 2: ResNet-32 accuracy with CIFAR-10 (a) and ILSVRC-2012 (b) bearing different noise patterns.

To study the two questions in this section, we first verify that a small number of key values in the input feature maps do affect the accuracy and they are termed as *sensitive values*. With just a little noise added, these sensitive values will have a great impact on the prediction accuracy. This means improper quantization of these values might seriously deteriorate the overall NN accuracy. In addition, we show that these values tend to aggregate in space in the various levels of input feature maps that form the *sensitive regions*.

As a result, sensitive values and regions should be carefully identified and quantized in fine-grained manner to reserve more detailed feature characteristics for the NN, while there is no such need for other insensitive regions. A natural implication is that we should take advantage of the sensitivity to speed up the convolution by differentiating quantization on different regions in the same feature map with almost no accuracy loss, which motivates our algorithm and architecture as elaborated in Section III and Section IV.

A. Sensitive Values

In this section, we will study the existence of input sensitive values in different datasets that affect the NN accuracy and to what degree. We take two widely used datasets—CIFAR-10 [19] and ILSVRC-2012 [20] on ResNet-32 as case studies.

First, we classify the input feature map values into several segments according to their magnitudes. For example, for layer i ($i = 0, 1, \ldots, 31$), we classify the values into three segments by using two thresholds that locate at 20% and 80% of the value distribution. That means segment 0 contains the largest 20% values of the feature map, segment 1 contains the middle 60% and segment 2 contains the smallest 20% data values. We then add noise onto the values in different segments. We use a constant factor u to represent the magnitude of noise. By changing u, we can adjust the intensity of the noise. Finally, we measure the NN accuracy and study the sensitivity of different segments by varying u. Fig. 2 gives the results, where we have several curves representing different patterns. For example, curve labeled as "TFF" represents noise being

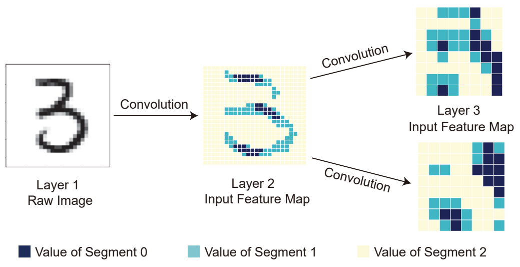


Fig. 3: Visualizing the sensitive regions in the feature maps for different layers of the three segments.

added only to segment 0, while curve "FTT" represents noise being added to both segment 1 and segment 2.

From the plot, several observations can be made:

1) When we add noise to only one segment, "TFF" curve drops more rapidly at the smaller u compared with "FTF" and "FFT";
2) As long as we add noise to segment 0, the network behaves approximately the same no matter whether we add noise to other segments because the curves "TFF", "TFT", "TTF" and "TTT" all coincide;
3) Segment 2 holding the small values in the feature map can tolerate larger noise as "FFT" only starts to degrade when u becomes very large.
4) The NN accuracy for different datasets has similar reaction to the noise. ILSVRC-2012 is slightly vulnerable to the noise than CIFAR-10 since it classifies thousands of object types that require more image details.

These observations clearly manifest that different input values (in segments) may have different impact (or sensitivity) on the final accuracy, and it is a popular observation across different networks we have investigated. To be specific, values in segment 0 are more sensitive to noise than other segments, which implies that the largest set of values in the input feature map are relatively more sensitive to the accuracy. In other words, we can retain the NN accuracy for a given dataset by restricting the quantization impact (equivalent to noise) for the sensitive values in the feature maps, while quantization for the insensitive values can be relaxed.

B. *Sensitive Regions*

In this section, we will explore how these sensitive values are spatially distributed in the feature map. We take the LeNet-5 [21] with MNIST dataset [7] as a simple yet visualize-able example.

After training the network with MNIST, we randomly select a raw image, e.g. the number "3" to conduct the inference process. Fig. 3 visualizes the input feature maps for the first three layers. We also use different colors to mark the values into three segments. From the plot, we see that larger values in segment 0 do not randomly scattered in the input feature map across all layers. Instead, they tend to aggregate in space forming several sensitive regions. Meanwhile, smaller values in segments 2 and 3 dominate the feature map as insensitive regions.

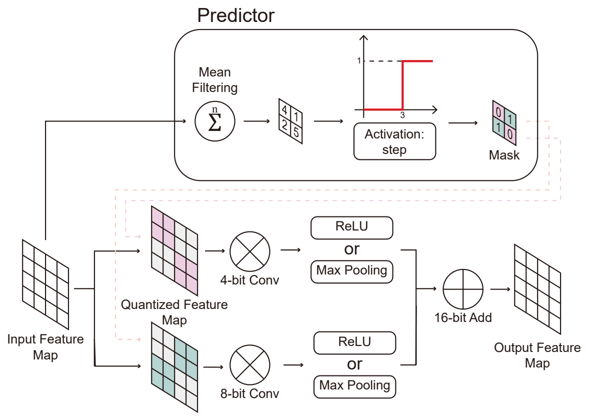


Fig. 4: DRQ algorithm overview.

We can apply a $x \times y$ rectangle to partition the whole feature map into individual regions, and each can be attributed as sensitive or insensitive. In turn, we can apply dynamic quantization policies for different regions. Given the observation that sensitive regions only account for a small percentage and insensitive regions dominate the entire feature map, the sensitivity analysis offers us a new opportunity to improve the inference efficiency with runtime quantization strategy.

Therefore, we propose a dynamic feature map region-based quantization—a software and hardware co-design scheme, to accelerate inference performance with nearly no accuracy loss for NN. The scheme consists of a DRQ algorithm and the corresponding architecture design.

III. Algorithm for Dynamic Region-based Quantization

Section II has shown that there are sensitive regions in the input feature maps. Based on this observation, there are two more problems need to be addressed:

1) How to identify the sensitive regions in the input feature maps at runtime. This process needs to be efficient and hardware friendly. Different from the weights that can be learnt offline, the input feature maps are not available until run time, and their sensitivity has to be extracted efficiently by the algorithm and hardware.
2) How to conduct an efficient input sensitivity aware convolution. Different sensitivity regions may introduce values with different precision in the same feature map with different quantization schemes. This intra-layer and multi-precision quantization requires fine-grained control on the underlying hardware to reduce the convolution workload and benefit the inference performance.

A. *Algorithm Overview*

The DRQ algorithm can be depicted in Fig. 4. The algorithm is consisted of two steps:

First, we design a prediction algorithm to locate the sensitive regions in the input feature maps. It performs mean filtering over the target feature map. Then we use step activation function to generate a binary mask, which can distinguish the

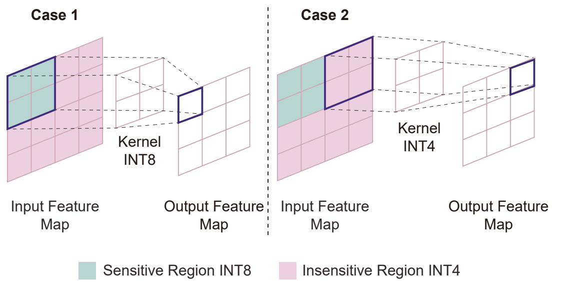


Fig. 5: Two cases of mixed-precision convolution, kernel over a sensitive region (left) and a insensitive region (right).

sensitive and insensitive regions in the input feature map. We will provide more details in Section III-B.

Second, we propose a mixed-precision convolution for inference computation, which adjust the kernel precision at runtime based on the sensitivity of the input feature map. For example, when the kernel slides over the sensitive regions (denote as the green blocks in Fig. 4), the convolution enters into high-precision mode with finer quantization of both weights and input feature map. Otherwise the convolution uses a low-precision mode over the insensitive regions (denote as the pink blocks in Fig. 4). We will provide more details in Section III-C.

B. Sensitivity Prediction

Given an input feature map of $h \times w$ dimension, we first quantize the input feature map from FP32 to INT8. This is a typical step in conventional quantization schemes because INT8 is sufficient for most inference workloads. Afterwards, we split it into several $x \times y$ regions. For each region, as shown in Fig. 4, we perform the dot-product with an all-one $x \times y$ kernel (mean filtering), which produces an output value. Consequently, we obtain $\frac{h \times w}{x \times y}$ output values. We compare all the output values to the predefined threshold. The comparison process can be treated as using the step activation function as shown in Fig. 4. The corresponding region is sensitive if the output value is larger than the threshold. We finally generate a binary mask map with a dimension of $\frac{h \times w}{x \times y}$, where ”1” in the mask map indicates that the region is sensitive, while ”0” indicates that the region is insensitive. This can be applied to each input channel for sensitivity prediction.

C. Mixed-Precision Convolution

Once we distinguish the sensitive and insensitive regions, we perform the mixed-precision convolution.

Without loss of generality, in this paper, we define convolution with INT8 as high-precision and INT4 as low-precision. With the sensitivity information recorded in the mask maps, there can be two common cases for the mixed-precision convolution as shown in Fig. 5. Note that we always store the kernel weights in INT8 format in the DRAM. But the value of the input feature map can be stored in either INT8 or INT4 format depending on its sensitivity. In case 1, the region is sensitive. When the kernel slides over this region, we use INT8 to represent the weights and the values in the region and perform the 8-bit convolution. In case 2, the region is insensitive and the values in the region have been stored in the INT4 format in DRAM. When the kernel slides over this region, we clip the precision of the kernel weights to INT4 and accordingly perform the 4-bit convolution. We will give the detailed hardware support in Section IV-C.

However, the precision switching between regions of different sensitivities may complicate the convolution process, which will be discussed in Section IV.

D. Design Space Exploration

Obviously, the threshold determines the number of the sensitive regions. And consequently, the smaller number of sensitive regions, the larger number of INT4 computations that can speed up the convolution, but may affect the NN accuracy.

The size of the sensitivity region also matters. Different NN models may require different region sizes depending on the feature characteristics. In general, a larger region means coarse-grained identifier, which is a hardware-friendly choice but at the cost of lower NN accuracy. Alternatively, a smaller region may reserve fine-grained characteristics with higher NN accuracy, but may cause frequent precision switching and more complicated control, which can be seen in Section IV-C. Moreover, the region size also affects the efficiency of sensitivity predictor as presented in Section IV-E.

To ensure the NN accuracy, we need to retrain the predefined NN model and fine-tune the weights so that they can be aware of the sensitivity information in the feature maps. During this process, we can also find the proper threshold value and region size. First, we acquire the value distribution of the input feature maps in each layer. Based on the value distribution, we choose an initial threshold and a region size by empirically starting from some large values. Then, we retrain the model for guaranteed accuracy, during which we will apply the mix-precision convolution in the forward propagation, but full-precision backward propagation for weight updating to minimize the loss of NN accuracy. We repeat the above training process until the NN converges. Next, we conduct the mix-precision convolution for inference and evaluate whether the NN accuracy can meet the expected requirement. If yes, the threshold and region size are determined. Otherwise, we will repeat the above steps by halving the region size or threshold. Although trial-and-error, the above process can always find the satisfactory values within a few iterations.

IV. Architecture for Dynamic Region-based Quantization

Section III introduces the DRQ algorithm, which offers a new opportunity to reduce the computation workload with sustained NN accuracy by identifying the sensitivity regions in the feature maps. In this section, we will introduce the accelerator architecture to support the algorithm.

A. Architecture Overview

Conventional accelerators [26], [29] cannot support the DRQ algorithm due to two major challenges: 1) the DRQ algorithm requires dynamic mix-precision convolution depending on the input feature map values, which cannot be learned offline like weights. This requires the architecture to be capable of handling random precision switching at any time; 2) different precision levels can be interleaved inside a single PE array for convolution with different computing and memory throughput. Both problems call for a novel redesign of the accelerator architecture.

To this goal, the proposed DRQ architecture is illustrated in Fig. 6. It consists of several processing element (PE) pages, which communicate with memory through a global buffer. Each PE page mainly consists of a line buffer, a mixed-precision convolution array, an accumulation unit, an output buffer, an activation and pooling unit, a sensitivity predictor and the associated control logic.

The DRQ architecture can be briefly described as follows. The global buffer stores the raw input feature maps and weights. To feed the convolution array, there is an im2col/pack engine in each PE page to pull feature maps from global buffer, transform them into the regulated format and pack them into the input line buffers. It also works for weights by loading and fixing them into each column of the array. One of the key parts in each PE page is the convolution array supporting mixed-precision convolution which will be elaborated in Section IV-C. Partial sums are accumulated in the output buffers which also works for variable kernel sizes. After the activation and pooling function, the output feature maps are fed to another key part—the sensitivity predictor to identify the sensitive regions that can guide the DRQ for the next convolution layer. The sensitivity predictor can be implemented together with the pooling unit. Finally, the output results are stored into the global buffer.

As an accelerator customized for DNNs, there can be no data communication between PE pages if well scheduled. For example, we can keep the kernels of one filter in a PE but split the filers into different pages to eliminate the communication.

B. Line buffer and Input Data Packing

The line buffer lies between the global buffer and the convolution array. To start the convolution, in each PE page, the input feature maps are read from the global buffer. The input feature map values are transformed to fit the mixed-precision convolution, and packed with the binary masks into the line buffer. Then the input feature map values are fed to the leftmost PEs in the convolution array.

To match the systolic array convolution, the feature map stored into line buffer are transformed like the *im2col* function in GPUs, as shown in Fig. 7(a). Specifically, for every kernel on the feature map, it will be aligned into a column, and the column is staggering arranged into the line buffer. Note that every line in the line buffer lags one cycle after the above lines, so as to form the data shifting and partial sum accumulation in the systolic array [18].

To cater for the mixed-precision convolution, the input feature map are densely packed for a higher utilization of the

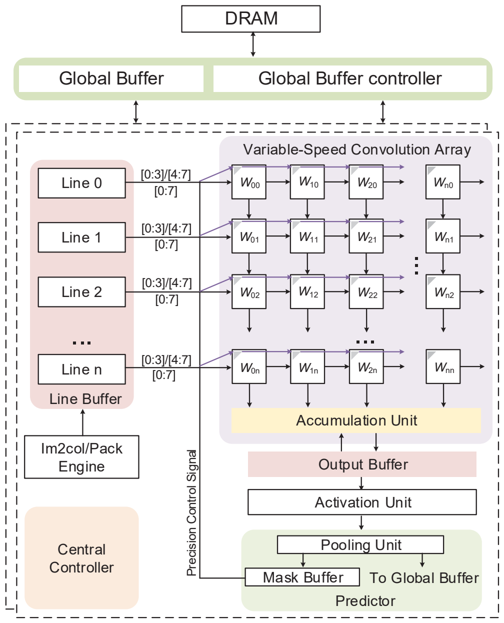


Fig. 6: Overview of the DRQ architecture.

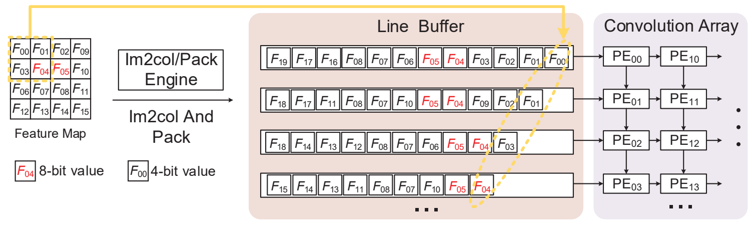
line buffer storage. The insensitive value is packed into a 4-bit register, like F_{00}, while the sensitive value is packed into a 8-bit register like F_{04} in the Fig. 7(a). The sensitivity of values can be extracted from the associated binary mask map accordingly. The binary mask from mask buffer will decide the working mode for the PEs in the convolution array.

C. Convolution Array with Variable Speed

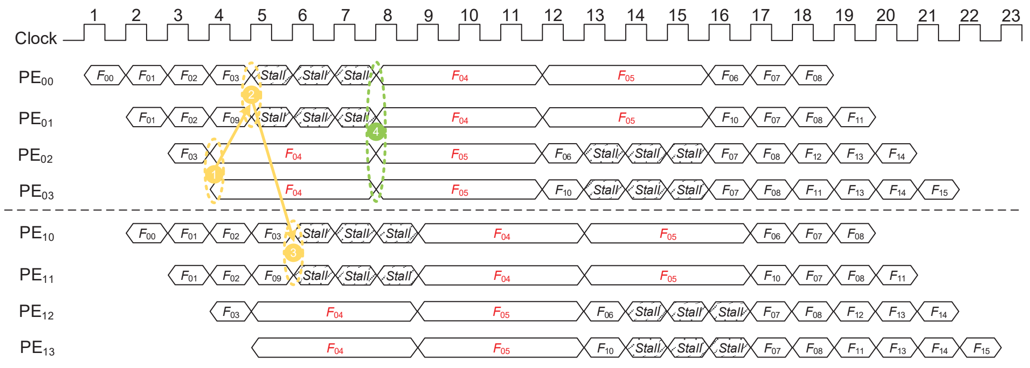
As shown in Fig. 6, the convolution array consists of PEs capable of mixed-precision computation. In DRQ architecture, the weights will always be quantized into INT8 format, and they are held into each PE in the array during the convolution process. The leftmost column of PEs accept new feature map values shifted from the line buffers for every clock cycle. Feature map values are shifted right and the partial sums are shifted down to the neighboring PEs every cycle.

1) Multi-Precision PE: The PE in the convolution array features by its multi-precision capability for MAC operation. In this study, we design our PE that is capable of executing two different precision (INT4 and INT8) because our DRQ algorithm proves that the two-level quantization is sufficient for most NN models without degrading the NN accuracy. Such a PE design can be found in previous work [29] exhibiting as an INT4 MAC, but can be united into an INT8 MAC taking four cycles in a timing-multiplexing manner.

Fig. 8 illustrates how to adapt the PE in our variable-speed systolic array. There are two 8-bit registers W and F holding a weight value and an input value, respectively. A 16-bit register P store the partial result. In default, the PE is in INT4 mode, whose INT4 MAC is used for the multiplication of a 4-bit weight with a 4-bit input. On demand, the PE can switch to INT8 mode depending on the binary mask map generated in the previous sensitivity prediction. To be specific, in cycle t,



(a) The Data Arrangement for Mixed-precision Convolution



(b) The execution timing for Mixed-precision Convolution

Fig. 7: The data arrangement for mixed-precision convolution (a); The execution timing for mixed-precision convolution (b).

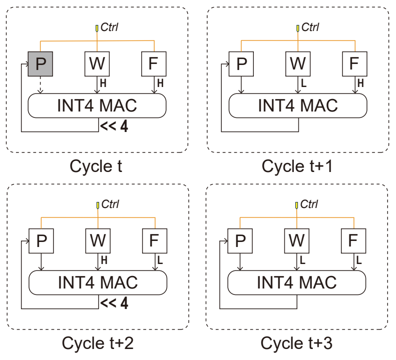


Fig. 8: The proposed multi-precision PE.

it extracts the higher 4 bits (H) of the weight value and the input value from W and F, respectively. The result is then shifted left by 4 and stored into the P register. In cycle $t+1$, it extracts the lower 4 bits (L) of weight value and the higher 4 bits (H) of input value from W and F, respectively for another 4-bit MAC accumulated with the previous product. The result is written into the P register. The behavior in cycle $t+2$ and $t+3$ is similar with t and $t+1$.

2) Execution Flow for Mixed-Precision: The convolution array is built from the dual-mode PEs. If all PEs get insensitive values, they behave as an INT4 unit. With any sensitive value shifted in, the convolution array can still work in a systolic manner but with a variable speed as explained in Fig. 7(b).

For example, the first column in the convolution array containing PE_{00}, PE_{01}, PE_{02} and PE_{03} will normally receive INT4 feature map values as shown in the first three clocks. However in the 4th clock, both PE_{02} and PE_{03} receive INT8 values because the convolution enters a sensitive feature map region, while PE_{00} and PE_{01} still receive INT4 values because their corresponding feature map values are insensitive. For MACs in the INT8 mode, PE_{02} and PE_{03} need four cycles to complete the 8-bit computation. To match the rate in the systolic array, PE_{00} and PE_{01} need to be stalled for three cycles waiting for PE_{02} and PE_{03} to complete. Consequently, the INT8 result is fully generated in four cycles. In cycle 8, PE_{00}, PE_{01}, PE_{02} and PE_{03} all receive INT8 values meaning they enter the sensitive regions, so all the four PEs will consume four cycles for computation. In cycle 12, a similar situation happens as in cycle 4, but this time PE_{02} and PE_{03} need to be stalled. Starting from cycle 16, all the PEs leaves the sensitive regions, so the systolic array returns back to the INT4 mode and can consume one new input value for every cycle.

For the second column in the convolution array containing PE_{10}, PE_{11}, PE_{12} and PE_{13}, it simply receives values shifted from the first column including the "stalls" control signal, so as to replicate exactly the same behavior as in the first column but with one cycle latency.

As seen from Fig. 7(b), our proposed mixed-precision convolution array can skillfully support variable-speed convolution, meeting the different requirements for sensitive and insensitive regions. It works at full speed in INT4 mode, and switches to INT8 computation on demand by inserting stalls.

3) Control Scheme for the Array.: The control scheme for the mixed-precision convolution array can be easy to implement following a similar systolic manner like the data flow, as shown in Fig. 7(b). In brief, normally all PEs work in the INT4 mode. Any PE receiving INT8 feature map value (identified by the binary mask) switches this column to INT8 mode (denote as signal ❶ in the plot). Those PEs have to spend four cycles to finish the computation (denote as signal ❹). Meanwhile, if there are any other PEs receiving INT4 on the same column, those PEs need to be stalled for three cycles (denote as signal ❷), and this makes all the PEs on the same column to synchronize in the systolic execution. In addition, the "stall" signal will be shifted to the right-neighboring column in the array with one cycle latency (denote as signal ❸), making the whole array still work in a systolic manner facing inputs with different sensitivity.

The control signals will not only control the timing of the array, but also serve as the local control for each PE as in Fig. 8, where the PE can be set to work in either INT4 or INT8 mode depending on the control signal.

D. Output Buffer and Accumulation Unit

The output buffer lies between the convolution array and the predictor. It temporarily stores the partial results and performs an in-place accumulation for the output feature maps. At the same time, it feeds the output feature maps to the activation/pooling unit, whose results are send to the sensitivity predictor to quantities the output feature maps with the binary masks. In our design, the traditional "activation-pooling" process in NN needs to be extended with one more step as "activation-pooling-prediction". Since the convolution process dominates the execution time, the added "prediction" step can be well inserted into the existing NN pipeline, with negligible performance overhead as described in Section IV-E. We design a dual buffering scheme for the output buffer to parallelize the convolution accumulation and the "activation-pooling-prediction" process.

The proposed output buffer and accumulation units also support variable kernel sizes and channels. Our proposed convolution array prioritizes the kernel size of 3×3 as in most of the NN models, but also supports other configurations, such as 5×5 and 7×7 with the aid of the the accumulation unit. For larger kernel size, we will split the kernel into several sub-kernels and launch them to the convolution array to obtain the partial sum. Finally, we accumulate the partial sum of each sub-kernel in the accumulation unit to calculate the final result. This is a common practice widely used in systolic array based NN accelerators.

E. Sensitivity Predictor

The sensitivity prediction is the key to enable DRQ which differentiates it from conventional quantization techniques. Based on the predicting scheme discussed in Section III-B, which is actually a mean filtering, we propose a sensitivity predictor design that can reuse the pooling unit without adding much complexity to the inference process.

Since the current feature map undergoing pooling will naturally become the input for the next convolution layer, we can reuse the pooling results to serve the predictor. Consider a $x \times y$ prediction window containing pooling windows of size $n \times n$. For $x = y = 4$ and $n = 2$ as the example illustrated in Fig. 9(a), we can directly take the four average pooling results for prediction instead of adding them again. As shown

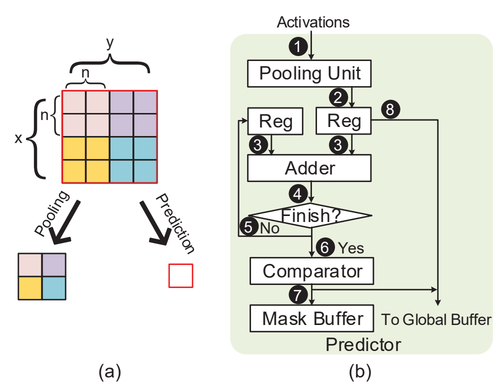


Fig. 9: The pooling and prediction process (a); The pooling unit and the predictor of the proposed architecture (b).

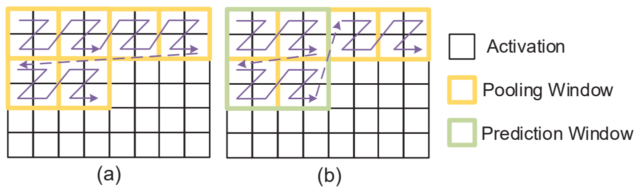


Fig. 10: Z-scan of activations during the pooling process (a); Z-scan of activations during the prediction process (b).

in Fig. 9(b), we first conduct the pooling on a window (denote as ❶). Then we store the pooling results into the temporal buffer (❷), which are then added with the partial result of the predictor (❸). Next, we check whether all the pooling windows in the prediction window have finished (❹). If no, we keep the partial prediction result in a register waiting for the next pooling (❺). If yes, we compare the final prediction result with a predefined threshold (❻), which actually generates the binary mask map (❼). The binary mask also indicates how the pooling results in the temporal registers should be quantized (INT8 or INT4). The binary mask is stored back to the mask buffer to assist the feature map for the next convolution layer (❽).

As illustrated in Fig. 10 (a) and (b), the access order of activations in the pooling and prediction process are different. This is because the window sizes of pooling and prediction are different. Therefore, the pooling results need to be held in a temporal buffer for some time for the later computation in the predictor, which requires additional on-chip storage. In brief, we need to buffer $\frac{w}{y}$ partial prediction results and $\frac{w}{n} \times \frac{x}{n-1}$ pooling results, where w is the width of the feature map. From this, we can see that a stripe-shaped region (large y and small x) is more economic if not hurting NN accuracy and is more beneficial for storage saving as observed in our experiments in Section VI-B2. Note that the storage overhead of 4×16 region size is only 2KB in ResNet-18, which is negligible compared to the capacity of line buffer.

Given there can be other pooling configurations in different networks, such as the convolution layer is not followed by an average pooling or even other operations. We still need to

	Eyeriss	BitFusion	OLAccel	DRQ
# PEs	224	3168	2499 (2448+51)	3168
Bitwidth	INT16	INT4	INT4+INT16	INT4
area (mm2)	0.32	0.32	0.32	0.32
Global Buffer	5 MB			

TABLE II: Configurations of different accelerators (Unit area of INT4, INT8 and INT16 MACs are 100.5, 377.5 and 1423 um^2, under TSMC_45nm).

execute the above prediction shown in Fig. 9(b) except that we do not store the pooling results back to the global buffer. We need to keep the average pooling adders to work in parallel with other operations.

V. Experimental Methodology

A. Validation on NN Accuracy

To valid our DRQ algorithm with two-level (INT8 and INT4) quantization, we implement it in the Tensorflow framework [1]. To be specific, we built a computing graph containing INT4 and INT8 computation using the built-in API called and the binary mask map generated at runtime. Although INT16 is sufficient for widely used CNN networks for image classification [6], [13], [18] including AlexNet, VGG, and ResNet, recent TensorRT results [9] show that they can be quantized to INT8 without accuracy loss, so we did not apply retraining for the quantization to INT8. However, DRQ itself is not precision limited and it can be easily extended to support INT16 and above.

The datasets we used are CIFAR-10 [19] and ILSVRC-2012 [20]. Note that they two are representative and well-acknowledged in various CNN accelerator designs for image classification [16], [26], [29]. The NN models we used are AlexNet, VGG16, ResNet-18, ResNet-50, Inception-v3 and MobileNet-v2, which cover a wide range of classic and modern networks with different parameter sizes.

We compare the NN accuracy of DRQ with the following designs: 1) Eyeriss [5], which is a spatial energy-efficient dataflow architecture. It uses coarse-grained quantization of INT16 throughout the network. We will use it as the baseline for standard NN accuracy; 2) BitFusion [29], which is an accelerator featured by the composable MAC unit. It employs the model typologies proposed in prior work [24], [35]. If the algorithm permits, it can change quantization at the layer granularity. If a coarse-grained quantization of INT16 is applied throughout the network, it can retain the accuracy as Eyeriss; 3) OLAccel [26], which is a state-of-the-art accelerator based on the outlier-aware low-precision computation. It does a static quantization on the weight values based on magnitude. For comparison, we take the results of AlexNet as reported in their paper [26]. For ResNet-18, ResNet-50, VGG16, Inception-v3 and MobileNet-v2 that are not available in their paper, we reproduce their experiments and report the results.

B. Modeling Accelerator Architecture

Table. II shows the configurations for all the accelerators in our experiments. We use 45nm TSMC technology library to study the area of the MAC units with different precision, where an INT16 MAC unit is almost 16X larger than an INT4 MAC unit. So, with similar area budget, Eyeriss has a total of 224 INT16 MAC units, BitFusion has 3168 INT4 MACs and OLAccel has 2448 ($= 3 \times 8 \times 6 \times 17$) INT4 MAC units and 51 ($= 3 \times 17$) INT16 MAC units. Accordingly, DRQ architecture has 3168 INT4 PEs (MACs) that are organized into 16 PE pages with 18 rows and 11 columns of PEs. Note that each PE can freely switch between INT4 or INT8 modes just as in BitFusion. The INT8 mode will have 1/4 the computing power of the INT4 mode. For a fair comparison, we use the same global buffer capacity and memory bandwidth for all these accelerators, and use CACTI [31] to estimate the area, power and latency to meet our design goal.

To evaluate the performance of our proposed DRQ architecture, we develop a cycle-accurate simulator to simulate the mixed-precision convolution array with line buffers and output accumulation together with the predictor. To be specific, we dump both the input feature maps and the binary mask maps for inference using Tensorflow, which are then fed into our simulator to get the cycle-accurate performance. Under 500MHz PE frequency, we verify that the required memory bandwidth is much smaller than the typical memory bandwidth provided by DDR3. So, with the regulated format of input data cached in the large global buffer, the algorithm can sustain a non-blocking convolution with multi-precision support.

VI. Experimental Results

A. Accuracy, Performance and Energy Consumption

Fig. 11 first shows the NN accuracy for the six networks with CIFAR-10 and ILSVRC-2012 datasets. For example, regarding ResNet-50, our DRQ algorithm shows nearly no accuracy loss for CIFAR-10 compared to Eyesiss (full INT16) and BitFusion (full INT8), and a 1.5% accuracy improvement over the OLAccel algorithm. For ILSVRC-2012, DRQ algorithm shows a 0.9% accuracy loss compared to Eyesiss and BitFusion, and a significant 5.1% accuracy improvement over OLAccel. This is because DRQ algorithm can identify the sensitive regions in the input feature maps on the run and fortify the computation precision over those regions. Instead, OLAccel only statically analyzes the numerical distribution of the weights and quantizes the weights accordingly, missing the real geometric feature implication on the feature maps. Therefore, for the sensitive regions in the feature map, DRQ can reserve more major feature information with high-precision (INT8) quantization.

Fig. 11 also shows the percentage of high- and low-precision used in the computation for the six networks. Both OLAccel and DRQ take full advantage of low-bit quantization and the majority of the computation is done with INT4 mode. However, OLAccel performs a static quantization so that the bit configuration for a certain NN is determined at pre-run time and fixed across all the layers and datasets. However, in DRQ, the quantization is dynamically decided according to the feature map predictor, so that the number of bits used for weight and activation values may vary from time to time. As demonstrated in the up-to-date Inception-v3, DRQ (0.7%

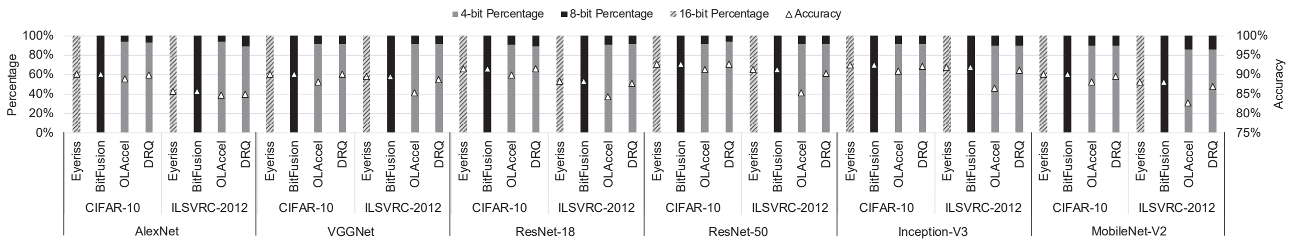


Fig. 11: Comparison of NN accuracy and percentage of 8/4-bit for different networks.

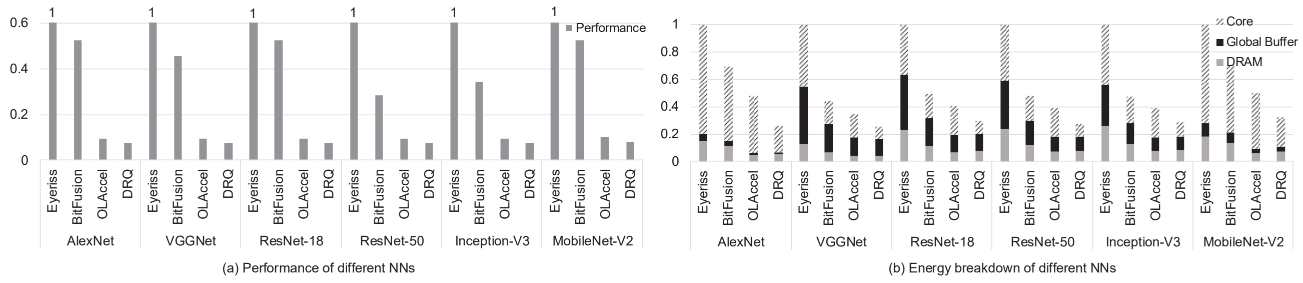


Fig. 12: Performance (a); energy breakdown (b) of different networks.

loss) is better at preserving accuracy than OLAccel (5.1% loss) with a similar 90% of 4-bit percentage. While for the more compact MobileNet-v2 which has a much smaller parameter size, the shifting from weight quantization to input feature map quantization makes our DRQ more robust for preserving NN accuracy (0.9% loss) than OLAccel (5.4% loss) with 85% of 4-bit percentage.

Fig. 12(a) shows the total execution cycles on different accelerators for the six networks, which are all normalized to Eyeriss. Still taking ResNet-50 as an example, compared to Eyeriss, DRQ achieves nearly 93% performance improvement because DRQ mostly uses INT4 MACs for insensitive regions with only a limited number of INT8 MACs for sensitive regions. About 16 times larger number of INT4 MAC units can be fit into the same area budget containing INT16 MACs used in Eyeriss. Compared to BitFusion, DRQ achieves 73% performance improvement because BitFusion mainly utilizes INT8 for computation in the comparison. For OLAccel, since it applies dedicate INT16 MACs for the first layer and separate INT4 MACs for the rest layers [26], DRQ can achieve 20% performance improvement thanks to the larger number of PEs under the same area budget in our scheme. Note that the speedup brought by OLAccel is at the cost of about 5% NN accuracy loss for benchmark like ILSVRC-2012.

Fig. 12(b) shows the energy consumption of different accelerators for the six networks, which is decomposed into DRAM, global buffer and processing cores (Core). Specifically, for ResNet-50, compared to Eyeriss, BitFusion and OLAccel, DRQ consumes 72%, 43% and 32% less energy, respectively. The energy reduction against Eyeriss and BitFusion is mainly due to the reduced precision on PEs, and the resultant narrower bit-width data transferred between DRAM and global buffer. Compared to OLAccel, energy consumption diversifies in different components. DRQ consumes more energy from DRAM

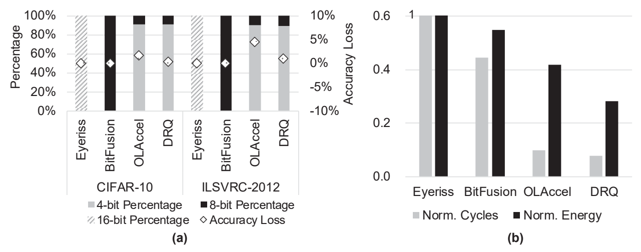


Fig. 13: Average accuracy loss (lower is better) (a), average normalized performance and energy (b).

because all weights are kept as INT8 in DRAM in DRQ, while most of the weights are set to INT4 in OLAccel. We choose the weight stationary scheme so that we can avoid frequently reading INT8 weights from global buffer because the weights are kept in the systolic array. Consequently, DRQ consumes similar energy as OLAccel on global buffer. The reduced core energy in DRQ is mainly due to the systolic array processing of the convolution so that most of the data can be shifted from neighboring PEs. However, to make OLAccel viable, their architecture is designed more towards a GPU processing style requiring each PE to fetch weight and activation from the local register file every cycle so as to cause a large amount of data accessing energy. Overall, DRQ consumes less energy than OLAccel on the core computation part.

Fig. 13 reports the average accuracy loss, execution cycles and energy across the six networks. Note that in this plot, we show the accuracy LOSS (lower is better). On average, for CIFAR-10, DRQ incurs 0.3% accuracy loss compared to Eyesiss and BitFusion, while OLAccel has 1.6% accuracy loss instead. For ILSVRC-2012, DRQ incurs 0.8% accuracy loss compared to Eyesiss and BitFusion, while OLAccel has 4.3% accuracy loss, which might be unacceptable for certain

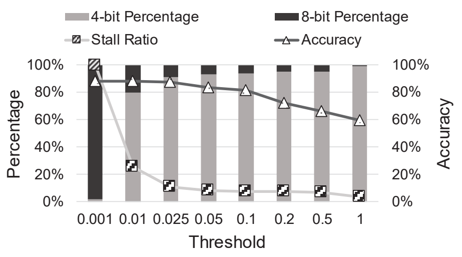


Fig. 14: Analyzation of the threshold.

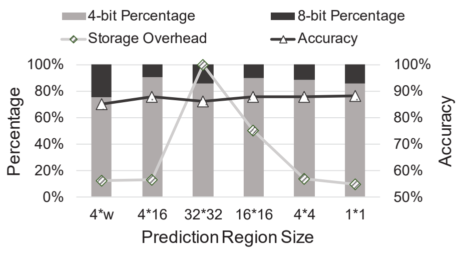


Fig. 15: Analyzation of the sensitivity feature map region size.

NN Model	Region Size	Average Threshold in Integer Number
AlexNet	2*4	18
VGG16	2*4	17
ResNet-18	4*16	21
ResNet-50	4*8	19
Inception-v3	4*8	23
MobileNet	2*4	25

TABLE III: The region size and the average threshold used.

applications. In terms of performance, DRQ improves the performance by an average of 92% and 83% compared to Eyeriss and BitFusion, and 21% compared to OLAccel with 3.3% accuracy improvement at the same time, which is quite impressive for inference. Meanwhile, DRQ reduces 72%, 49% and 33% energy consumption on average over the other three schemes. Note that DRQ has the capability of tuning the threshold value to sustain the same accuracy levels as Eyesiss and BitFusion, while gaining notable performance.

In conclusion, by properly predicting the sensitive regions in the feature map and applying the sensitivity-aware quantization algorithm, the dynamic mix-precision convolution array can reserve much of the key feature characteristics to deliver a faithful NN accuracy comparable to the full-precision model, while at the same time saving much of the bandwidth and computing power for the dominant insensitive regions.

B. Design Space Exploration

As discussed in Section III-D, the threshold and the region size affect the performance of our proposed architecture and the NN accuracy. In this section, we want to determine the appropriate threshold by balancing the relationship among the 4-bit percentage (higher is better), the stall ratio in the systolic array (lower is better) and the NN accuracy (higher is better). Based on the determined threshold, we choose the suitable region size by making tradeoffs among the 4-bit percentage (higher is better), the storage overhead (lower is better) and the NN accuracy (higher is better).

1) Impact of Threshold: Fig. 14 shows the impact of the threshold value for ResNet-18. We sweep it from 0.001 to 1. Generally, a higher threshold means less sensitive regions and allows for more INT4 quantization, but it will degrade the NN accuracy. Moreover, less sensitive regions lead to lower stall ratio. Note that the stall ratio represents the probability of bubbles in the realtime execution. From this plot, we find that with a threshold of 0.025, DRQ achieves an optimal point for ResNet-18.

2) Impact of Region Size: Fig. 15 studies the impact of region size on ResNet-18 considering the 4-bit percentage, storage overhead for buffering used in the predictor and NN accuracy. We set the region sizes to $4 \times w$, 4×16, 32×32, 16×16 and 4×4, where w is the feature map width. From the plot, we see that different region sizes may have different impacts. For a too small region size like 4×4, the sensitivity of a region can be easily affected by the strong noise in one pixel, leading to degraded accuracy and therefore requiring more INT8 quantization to recover the accuracy impact. For too larger region size like 32×32, most of the regions in the feature map have a high chance to cover some part of feature characteristic, ending up with many regions unnecessarily being marked as sensitive incurring redundant INT8 quantization. Furthermore, as discussed in Section IV-E, the region size also affects the storage overhead for prediction. In Fig. 15, we normalize the storage overhead to that in the 32×32 case. We find that the stripe-shaped regions like $4 \times w$ will be beneficial for the storage saving in ResNet-18. Specifically, the plot suggests that a 4×16 region size is the best choice for balancing all above factors.

Note that the optimal threshold and region size differ from network to network. Table III shows the actual region sizes and thresholds we use for all the investigated networks. The thresholds are set to different integer numbers for different layers, so we only show the value averaged across all the layers in the table. We can find the most suitable settings through an offline study, while the architecture can set the corresponding parameters at run time.

We need to mention that as the layers go deep, the size of the feature maps shrinks due to the pooling operation and we need to scale the region size accordingly. For the last a few convolution layers, the size of the sensitivity region is reduced and fixed at 2×2 in our experiments. As for the threshold, it remains similar in the front layers and may become 5X smaller in the last few layers. This is because most of the activations aggregate as small values towards zero.

C. Performance Breakdown

For a better understanding of the DRQ, we make a detailed performance breakdown on ResNet18.

Fig. 16 shows the execution cycles for different algorithm blocks (C1, B1, B2, B3 and B4) in ResNet-18, which are normalized to the total execution cycles. Note that a block in ResNet-18 contains multiple layers. From the plot, we can see that different blocks occupy different proportions of

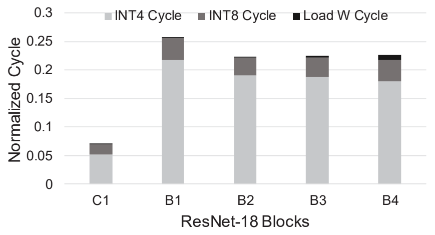


Fig. 16: Utilization breakdown.

execution time. No matter in which block, the computation cycles dominate the data loading cycles. Due to the different ratio of sensitive regions in different blocks, the INT4 and INT8 execution cycles change across blocks. It is worth noting that the first block (C1) of ResNet-18 is more sensitive than all other blocks, so the INT8 operation ratio is higher (12% of total execution cycles) in the first block.

The DRQ is idling when switching to another set of weights (Load W). Because the size of the feature maps is much smaller for the last block (B4), the weight loading process may account for relatively longer period to around 4% of the total execution cycles in that block. For other blocks, the weight loading overhead is negligible.

VII. Background and Related Works

A. Neural Network Accelerators

In the last few years, various NN accelerators are proposed for inference [4], [6], [8], [11], [13], [18], [28], [30], [36]. Some of the representative designs and techniques are,

1) Diannao series—on-chip memory design: Diannao series [6], [8], [23] introduce multiple NN accelerators. DaDianNao [6] integrates a large on-chip eDRAM to avoid frequent off-chip memory accessing. ShiDianNao [8] accelerates small NN by storing all parameters of NN into a small on-chip SRAM. On top of these accelerators, Cambricon [23] is designed as a domain-specific instruction set architecture to support multiple kinds of NN models. Since reducing off-chip memory accesses is important to performance, our DRQ architecture also benefits from these on-chip memory designs.

2) Eyeriss—dataflow analysis: As large NNs are impossible to fit the on-chip memory, researchers proposed dataflow analysis. Eyeriss [5] explores input-stationary (IS), weight-stationary (WS), output-stationary (OS), and no-local-reuse (NLR) and propose row-stationary (RS) dataflow to improve data reuse. Nowadays, dataflow analysis becomes essential for NN accelerators. Our DRQ architecture supports IS, WS, OS and RS, but applies WS in priority because the storage overhead of weights is larger than input values.

3) TPU—systolic array: Systolic array in TPU [18] is efficient for convolution, as it can reuse data as much as possible. Our DRQ proposal steps forward by closely interacting with the algorithm, and leverages the systolic array as an efficient infrastructure for mixed precision.

4) Accelerators for NN pruning and quantization.: General NN pruning and quantization are widely used for inference. To be specific, pruning makes NN sparse, quantization reduces NN precision and both reduce the required memory bandwidth. EIE [14] and Cnvlutin [2] exploits sparse NN models, while their special data format requires additional hardware overhead. BISMO [32] and BitFusion [29] proposed to execute quantized NN models. OLAccel [26] is the most up-to-date accelerator that utilizes quantization with 4-bit and 16-bit MACs. For the first layer, OLAccel applies 16-bit MAC, while for the rest layers, it applies 4-bit MAC. These designs mainly cater for general NN pruning and quantization. Consequently, their architectures struggle for peak performance because it is difficult to find a perfect fitted quantization algorithm.

B. Quantization algorithms

Quantization algorithms compress the NN model by reducing the number of bits required for weight and input feature map value. Some of the representative algorithms are,

1) Weight clustering: Gong et al. [12] and Wu et al. [34] applies k-means clustering to reduce the number of values that weights can represent. Han et al. [15] quantizes weights using weight sharing and then applies Huffman coding to the quantized weights for higher compression rate. However, the weight clustering methods require special data indexing, which is time-consuming and unfriendly to hardware.

2) Binarization: For higher compression rate, several works are proposed to quantize NN by binarized or ternary weights and input feature map values. Lin et al. [22] binarizes the weights and input feature map values into $\{-1, +1\}$. Zhou et al. [37] binarizes weights into $\{-w, +w\}$. Zhu et al. [38] maps weight into $\{-w_N, 0, +w_P\}$. However, these works achieve high compression rate at the cost of notable accuracy loss.

3) Intra-layer quantization: Considering the importance of NN accuracy, Park et al. [27] uses low-bit precision on the majority of small data while applying high-bit precision on a small part of large data. However, since the input feature map values are obtained on-line, quantizing input feature map cannot be supported by accelerators in [25], [26], [29], [32] due to the mismatch in hardware.

In summary, a close interaction between algorithm and hardware is still missing in above works, which we believe will be the trend for future NN accelerators. Our proposed DRQ makes such a trial to closely link them by respecting the NN accuracy and enabling higher accelerator performance.

VIII. Conclusion

Contemporary quantization algorithms by quantizing weight or activation into integer values only focus on the value magnitude statistics but fail to capture the inherent geometric characteristics in the input feature maps. This paper first studies the sensitive regions in the feature map that may affect NN accuracy. To reduce the computation complexity, we propose DRQ algorithm to dynamically quantize the convolution based on the sensitivity regions in the input feature map. Moreover, we propose the DRQ architecture to efficiently implement the proposed DRQ algorithm through a well-designed mixed-precision convolution array. Our evaluation shows that the proposed DRQ scheme outperforms other similar schemes in performance, energy or accuracy.

Acknowledgment

We thank the reviewers for their thoughtful comments and suggestions. This work is partly supported by the National Natural Science Foundation of China (Grant No. 61772331, 61972242, 61834006) and National Key Research and Development Program of China (Grant No. 2018YFA0701500). Naifeng Jing is the corresponding author.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News*, 44(3):1–13, 2016.

[3] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6541–6549, 2017.

[4] Srimat Chakradhar et al. A dynamically configurable coprocessor for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 38(3):247–257, 2010.

[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379, 2016.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[7] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[8] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104, 2015.

[9] Szymon Migacz et al. 8-bit inference with tensorrt, 2017.

[10] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225–2236, 2010.

[11] Vinayak Gokhale et al. A 240 g-ops/s mobile coprocessor for deep neural networks. In *CVPR Workshops*, pages 682–687, 2014.

[12] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[13] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[14] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

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

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[18] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[23] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. Cambricon: An instruction set architecture for neural networks. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 393–405, 2016.

[24] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

[25] NVIDIA Whitepaper. Nvidia turing gpu architecture, 2018.

[26] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 688–698, 2018.

[27] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 580–595, 2018.

[28] Maurice Peemen et al. Memory-centric accelerator design for convolutional neural networks. In *ICCD*, volume 2013, pages 13–19, 2013.

[29] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775, 2018.

[30] Vivienne Sze et al. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[31] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.1. Technical report, Technical Report HPL-2008-20, HP Labs, 2008.

[32] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Själander. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *IEEE 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 307–3077, 2018.

[33] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[34] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[35] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] Chen Zhang et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *FPGA*, pages 161–170, 2015.

[37] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9426–9435, 2018.

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