ENCORE Compression: Exploiting Narrow-width Values for Quantized Deep Neural Networks

Myeongjae Jang
KAIST
School of Computing
Daejeon, Republic of Korea
myeongjae0409@kaist.ac.kr

Jinkwon Kim
KAIST
School of Computing
Daejeon, Republic of Korea
coco@kaist.ac.kr

Jesung Kim
KAIST
School of Computing
Daejeon, Republic of Korea
jesung.kim@kaist.ac.kr

Soontae Kim
KAIST
School of Computing
Daejeon, Republic of Korea
kims@kaist.ac.kr

***Abstract*—Deep Neural Networks (DNNs) become a practical machine learning algorithm running on various Neural Processing Units (NPUs). For higher performance and lower hardware overheads, DNN datatype reduction through quantization is proposed. Moreover, to solve the memory bottleneck caused by large data size in DNNs, several zero value-aware compression algorithms are used. However, these compression algorithms do not compress modern quantized DNNs well because of decreased zero values. We find that the latest quantized DNNs have data redundancy due to frequent narrow-width values. Because low-precision quantization reduces DNN datatypes to a simple datatype with less bits, scattered DNN data are gathered to a small number of discrete values and incur a biased data distribution. Narrow-width values occupy a large proportion of the biased distribution. Moreover, an appropriate zero run-length bits can be dynamically changed according to DNN sparsity. Based on this observation, we propose a compression algorithm that exploits narrow-width values and variable zero run-length for quantized DNNs. In experiments with three quantized DNNs, our proposed scheme yields an average compression ratio of 2.99.**

***Index Terms*—DNN, Compression, Quantization, NPU**

I. Introduction

Deep Neural Networks (DNNs) become a practical machine learning algorithm running on Neural Processing Units (NPUs) such as Eyeriss [1], [2]. As DNNs go deeper, the size of DNN data, such as weights and partial outputs, enlarges. This causes performance overhead due to heavy computation and data traffic. As researches show that the lower precision datatype does not affect DNN accuracy much [3], quantization is used to convert DNN data to simple datatypes for faster DNN processing. Current NPU architectures commonly use various quantization techniques [4]–[6] with simple low precision datatypes. Simple datatypes enable a light-logic NPU circuit with less data traffic.

Although low-precision quantization reduces the DNN data size, there are still a large amount of output data that cannot be held in the NPU on-chip buffer [4]. To address this problem, several DNN compression algorithms are proposed [7]. Because the commonly trained DNN data follow a normal distribution [6], [8], many small values are converted to zero through low-precision quantization. This causes high DNN sparsity, and so zero value-aware compression algorithms, such as zero run-length encoding (ZLE) and compressed sparse column (CSC), can be used. However, nowadays several layers with low sparsity are found in the latest quantized DNNs. Zero value-aware compression algorithms suffer to compress these low sparsity layers of DNNs.

We find that there are many narrow-width values in quantized low sparsity DNNs. A narrow-width value is defined as a value in which upper half of the data bits is equal to 0 or 1. Before quantization, it is hard to find bit-level data similarity on conventional 32-bit floating point (FP32) datatype because of complex exponents and mantissa logic. When quantization is applied, small absolute values are biased to zero and a small number of discrete values [4]. These small number of discrete values occupy a large proportion of the DNN data because they show a normal distribution [5], [6]. Moreover, the number of the small discrete values increases during retraining with quantization. Especially, the small discrete values become narrow-width values since they can be represented by a few lower part of the data bits. Moreover, the increase of narrow-width values with quantization can occur across multiple layers.

Based on our observations, we propose ENCORE compression technique that exploits narrow-width values to compress quantized DNNs. ENCORE considers narrow-width values that are frequently found in quantized DNNs as a compression target. To achieve narrow-width value compression, ENCORE classifies DNN data into three types: zero, narrow-width, and incompressible values, and uses a 2-bit flag for each type. For narrow-width values, ENCORE stores only lower half of the data bits with a flag. Because the deleted half bits are all 0s or 1s, they can be restored by sign extension with the flag. Additionally, for zero values, ENCORE proposes variable zero run-length encoding (VZLE) to cover existing high sparsity layers more efficiently. VZLE dynamically changes zero run-length bits based on the current compressible DNN data sparsity. With three quantized DNNs, ENCORE shows an average compression ratio of 2.99, while previous compression algorithms fail to compress. ENCORE incurs only 0.05% area and 0.23% power overheads in comparison to EIE [9].

II. Related Works

Several studies attempt to reduce the DNN data size through compression. One well-known DNN compression scheme is pruning [4]. Pruning compresses a DNN model structure by skipping weights. Therefore, pruning requires retraining because weight skipping can affect the accuracy. We focus on

978-3-9819263-6-1/DATE22/©2022 EDAA

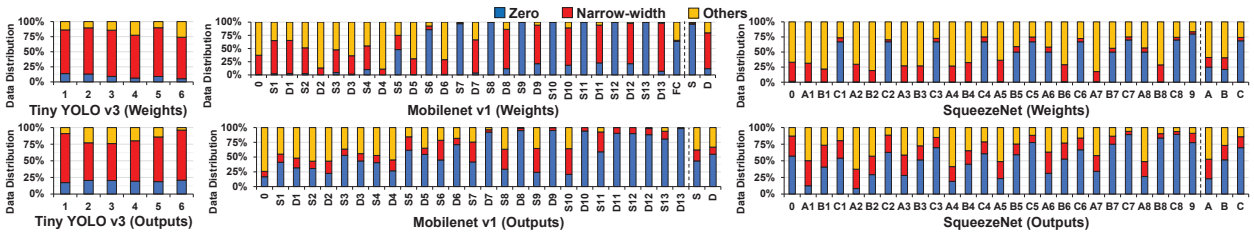

Fig. 1. DNN Weight (Up) and Partial Output (Down) Distribution of Tiny YOLO v3, Mobilenet v1, and SqueezeNet

data-level compression after training without changing a DNN structure; thus, pruning is orthogonal to our approach.

Compression algorithms are implemented as hardware. Several related works [10]–[14] propose compression algorithms for general purpose processes on cache and memory. Several DNN compression algorithms are implemented inside the NPU architecture. We summarize the previous compression algorithms for NPUs in Table I. RRAM-DNN [15] uses Huffman encoding [4] to DNN weights. Ko *et al.* [7] adopts the JPEG image compression to the pre-trained weight matrices. We group them as a weight-only compression because they only handle weight decompression in hardware. Because of the high complexity of the compression algorithms, weight compression should be performed by software after training. Moreover, they cannot support run-time compression, thus partial outputs are not supported. Because the current embedded systems cover training with frequently transferred inputs and outputs [5], [16], the shallow compression coverage is a serious defect. The weight-only compression also has an architectural overhead to handle metadata, such as a decoding table. Therefore, the weight-only compression is excluded from our experiments.

Another compression algorithm is a zero value-aware compression. A fixed ZLE is adopted on Eyeriss v1 [1], CSC on EIE [9] and Eyeriss v2 [2]. The fixed ZLE on Eyeriss v1 uses five zero run-length bits for compressing both DNN weights and partial outputs, and it can work on both training and inferencing processes. CSC inspects a 2D matrix by column and only stores non-zero values. The number of zero values between two non-zero values are encoded as a relative row index. CSC has the same coverage as the fixed ZLE for compressible DNN data types and processes. However, CSC requires more hardware modification than the fixed ZLE on both the processing elements (PEs) and on-chip buffer microarchitecture because CSC logic and results are more complex than the fixed ZLE. We use the fixed ZLE and CSC in our experiments as competitive compression algorithms because their compression coverage is appropriate to the current embedded systems.

III. Observation: Changes of DNN Sparsity

With the help of quantization, the sparsity of the latest DNNs has changed. In this section, we observe two submerged characteristics of the modern quantized DNNs: increased portion of narrow-width values and scattered zero values.

A. Lower Sparsity of Modern DNNs

We find that most low sparsity convolutional layers with optimized DNN structure tend to use smaller kernels than other layers. Tiny YOLO v3 [17] uses 1×1 or 3×3 kernels.

TABLE I
Comparison of DNN Compression Algorithms with ENCORE

Name	Weight-only [7], [15]	Fixed ZLE [1]	CSC [9]	**ENCORE**
Algorithm Complexity	Very High	Very Low	High	**Low**
HW Comp.	No	Yes	Yes	**Yes**
Target Data	W	W, O	W, O	**W, O**
Target Process	I	I, T	I, T	**I, T**
Compressible Data	Low Entropy Data	Zero	Zero	**Zero, Narrow-width**
Arch. Overhead	Medium	Low	High	**Low**
Modified Parts	Buffer	Nothing	PE, Buffer	**Nothing**

(W = Weight, O = Partial Outputs, I = Inference, T = Training)

Mobilenet v1 [16] utilizes depthwise separable convolution which separates one standard convolutional layer as a kernel-level depthwise layer ('D' in Figure 1) and channel-level separable pointwise layer ('S' in Figure 1). SqueezeNet [18] proposes an effective group of convolutional layers called a fire module. It consists of one squeeze layer and two expand layers with 1×1 and 3×3 kernels. 'A', 'B', and 'C' in Figure 1 represent the squeeze layer and the two expand layers, respectively. The distribution of partial outputs is related to activation function. Because Tiny YOLO v3 uses LeakyReLU [8], negative values remain unchanged, and the YOLO Layer follows for fine-tuning partial outputs. It affects the output distribution.

Figure 1 shows DNN data distribution of three modern DNNs. For the weight distribution, Tiny YOLO v3 and depthwise layers in Mobilenet v1 show extremely low sparsity, of which the zero value distribution does not exceed 20%. Squeeze layers in SqueezeNet also show low sparsity under the 25% zero value distribution. For partial outputs, Tiny YOLO v3 shows the zero value distribution of less than 20%. Average zero value distribution of Mobilenet v1 outputs do not exceed 50%. Therefore, we can infer that the conventional zero value-aware compression algorithms do not work well for these modern DNNs because of the lack of compressible zero values.

B. Increased Distribution of Narrow-width Values

Because modern DNNs have low sparsity layers, we try to find other compressible values. As low-precision quantization with small data bits is applied to modern DNNs, we find that many narrow-width values are generated within the DNN data. Before quantization, 32-bit floating-point (FP32) datatype values do not have bit-level similarity. Because most DNN data follow the normal distribution [5], [6], through low-precision quantization with fixed point or integer datatypes, a significant number of values are biased to zero and small absolute values. Biased small absolute values have bit similarity and become

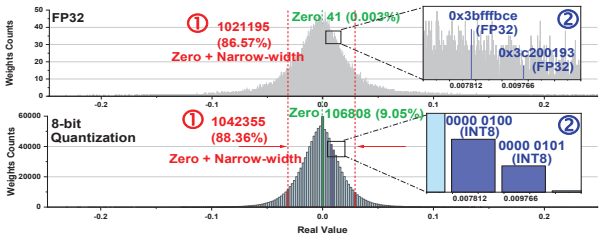

Fig. 2. Effect of Quantization on Narrow-width Values

narrow-width values, of which upper halves are equal to 0 or 1. Figure 2 is an example of 8-bit integer (INT8) quantization to FP32 datatype weights in the fifth convolutional layer of Tiny YOLO v3. First, scattered small absolute values are gathered to one discrete value through quantization (**Red** ① in Figure 2). On the FP32 datatype weight distribution, counts of any values do not exceed 50. This means that there are lots of floating-point values, but each of count for each floating-point value is small. Because FP32 is a high-precision datatype, a difference between two neighboring values is very small, less than 10^{-6}. After quantization, these two FP32 datatype values become the same INT8 value. Therefore, the weight distribution is biased, and nearly 90% of weights become zero or narrow-width values. Second, a quantized data bit pattern gives a chance to compress. As shown in **blue** ② in Figure 2, there are two continuous quantized narrow-width values, 0000 0100 and 0000 0101. Their quantized INT8 datatype bit patterns differ in only 1 bit to each other. However, FP32 datatype bit patterns for two values are 0x3bfffbce and 0x3c200193, respectively. Because of complex exponents and mantissa logic, it is impossible to find bit similarity even between close FP32 datatype values. Low-precision quantization makes the DNN data bit patterns simpler than FP32 datatype. Therefore, there are higher possibility to find bit similarity that can be utilized to compress on both data-level and bit-level.

We analyze the narrow-width value distribution of the three modern DNNs. According to Figure 1, Tiny YOLO v3 has about 60% narrow-width values for both weights and partial outputs. The depthwise convolutional layers in Mobilenet v1 ('D' in Figure 1) also contain 67.8% narrow-width values on average. The proportion of zero and narrow-width values are nearly equal in the squeeze layers and 1×1 expand layers in SqueezeNet ('A' and 'B' in Figure 1). The narrow-width values make up about 20% of the partial outputs of Mobilenet v1 and SqueezeNet. In brief, modern quantized DNNs contain several layers that have a large number of narrow-width values equal to or larger than zero values. If we compress both zero and narrow-width values, the compression coverage can be increased by at least 80%.

C. Scattered Zero Values

Even though we can compress the narrow-width values, there are still high sparsity layers. Moreover, zero value compression is still required to compress earlier DNNs. We further analyze zero values with the fixed ZLE and Figure 3 shows the compression ratio results. In this work, we define compression ratio as (*data size* ***before*** *compression*)/(*data size* ***after*** *compression*).

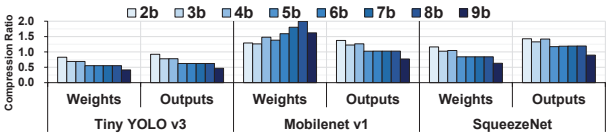

Fig. 3. Compression Ratio with Different Zero Run-Length Bits

A higher compression ratio indicates better compression with a smaller compressed data size. We find that the highest compression ratio for each DNN is achieved using different numbers of zero run-length bits. Compared to Figure 1, low-sparsity DNNs require a small number of run-length bits. There are fewer continuous zero values that can be dealt with by two or three zero run-length bits. Long zero run-length bits incur unnecessary overhead because several bits are never used to express short-length zero values. On the other hands, the 8-bit ZLE shows the highest compression ratio for Mobilenet v1 weights. This means that more than 128 continuous zero values are frequently found. With high sparsity DNNs, short zero run-length bits cannot hold continuous zero values longer than the maximum value that the run-length bits can express. For example, a 7-bit ZLE cannot express more than 128 continuous zero values at once. We need at least twice the run-length encoding. We define this situation as run-length bit overflow. The run-length bit overflow generates repetitive run-length bits and the related metadata overhead.

In summary, modern DNNs with different sparsities have different lengths of appropriate zero run-length bits. If we adjust the number of zero run-length bits dynamically during compression, a higher compression ratio can be achieved.

IV. ENCORE: Compression for Quantized DNNs

In this section, we present ENCORE compression technique and overall ENCORE-supported NPU architecture. ENCORE has two main design concepts as 'compressing narrow-width values' and 'variable zero run-length encoding (VZLE)'.

A. ENCORE Compression and Decompression

a) Compression: Figure 4 shows examples of the ENCORE compression. ENCORE classifies DNN data into four types and handles them with a 2-bit flag. The flag is concatenated to the front of the compressed DNN data. For narrow-width values, the upper half of the data is eliminated, and only the remaining bits are stored (**CASE 1**). As the example in Figure 4 assumes INT8 datatype, there is a 2-bit saved area after compressing the narrow-width values.

Continuous zero values are compressed using VZLE. **CASE 2-1** shows a compression example with 100 continuous zero values. The zero run-length bits start from three bits. They store the zero run-length one less than the actual run-length count. Therefore, initial three run-length bits can handle a maximum of eight continuous zero values. Because there are more than eight continuous zero values in this example, ZLE causes a run-length bit overflow. ENCORE increases the zero run-length bits by 1 bit whenever the run-length result reaches a maximum. For each increase, the run-length bits can hold twice the continuous zero values than before. The next zero run-length bits are 4 bits long and can hold up to 16 continuous zero values. To compress

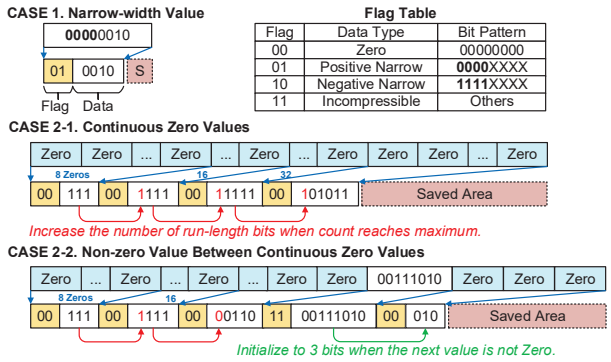

Fig. 4. ENCORE Compression Example

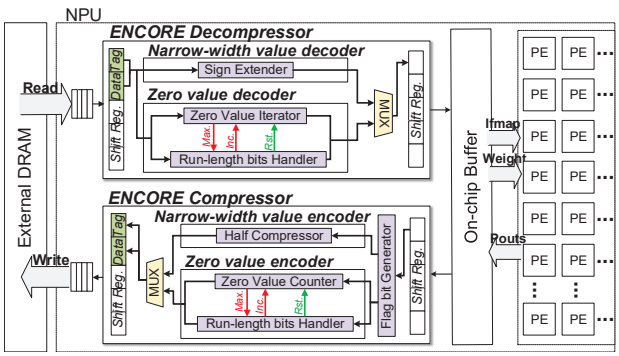

Fig. 5. ENCORE Architecture

100 continuous zero values, VZLE updates the run-length bits three times, and the final run-length bits are 6-bit long. The maximum number of variable zero run-length bits is the same as the datatype bits because several additional problems such as data misalignment can occur.

The increased zero run-length bits are reset to 3 bits when the next instance of DNN data is not zero: either a narrow-width or incompressible value (**CASE 2-2**). This case shows an example in which one incompressible value is located after 31 continuous zero values. First, continuous 31 zero values are compressed by VZLE with two run-length bit updates. Next, the incompressible value is found, and ENCORE resets the zero run-length bits to 3 bits. As a result, the last three continuous zero values are dealt with by three run-length bits. When there is a non-zero value, ENCORE assumes that the DNN data reach the low sparsity section with a small number of continuous zero values and directly resets long run-length bits to 3 bits.

As shown in Figure 3, VZLE can prevent many run-length bit overflows that can occur through Mobilenet v1 weights by using long run-length bits. Initial three run-length bits can achieve a reasonable compression ratio for low-sparsity DNNs such as Tiny YOLO v3. More zero run-length bits result in unused run-length bit overheads with a lower compression ratio.

b) Decompression: ENCORE decompression is performed by reversing the compression. ENCORE first reads two bits as a flag. Looking up the **Flag Table** in Figure 4, it computes the number of compressed data bits to read next. If the flag is '01' or '11', it reads half of the datatype bits and uses sign extension for upper half of the data (**CASE 1**) followed by the flag. For the '00' flag with VZLE, ENCORE first reads three bits and generates zero values repeatedly based on the read run-length bits. At the same time, ENCORE checks whether the zero run-length bits are all 1s. If they are, ENCORE increases the next run-length bits by one bit. Therefore, if the next flag is also '00', ENCORE reads one more zero run-length bit and repeats zero value generation. For **CASE 2-1**, ENCORE repeats this process three times with continuous '00' flags. Each compressed run-length bit is changed to 8, 16, 32, and 44 continuous zero values. ENCORE resets the next zero run-length bits to 3 bits if the read run-length bits are not all 1s. With the compressed result in **CASE 2-2**, ENCORE resets the updated number of zero run-length bits to read from 5 to 3 bits when it reads the zero run-length bits '00110'. The zero run-length bits, which are not the maximum values, mean that a non-zero value follows. Because most DNN data are stored as 1D vector or 2D matrix and read sequentially [1], ENCORE does not need to keep the number of updated zero run-length bits until reading the next flag bits.

B. ENCORE Architecture

ENCORE is designed as a hardware mircoarchitecture that can be imported to NPUs. ENCORE can compress and decompress both DNN weights and partial outputs at run-time. There are no limited processes to use ENCORE; thus, users can utilize ENCORE compression not only during the inference phase but also during the training phase. Figure 5 shows overall ENCORE architecture built in an NPU. ENCORE is located between the on-chip buffer and the external DRAM. By compressing DNN data evicted from the buffer, ENCORE increases DRAM bandwidth. Since ENCORE does not affect computation through PEs, any kind of NPU can adopt ENCORE without additional modification of the control or process parts. ENCORE obtains the DNN data as an 1D vector, and the given DNN data vector is temporarily stored in the shift register. For each DNN data, ENCORE compresses and decompresses according to the type of data: narrow-width or zero values. For the narrow-width values, ENCORE eliminates upper half of the data for compression. This can be restored using the sign extender during decompression. Zero values are dealt with the VZLE. The zero value counter in the ENCORE compressor counts the consecutive number of zero values and changes the count as the run-length bits. For decompression, the zero value iterator repeatedly generates zero values according to the run-length bits. The run-length bits handler manages the number of current run-length bits. When the zero run-length reaches a maximum, the handler increases the zero run-length bits by one bit. On the other hand, the zero run-length bits are reset to 3 bits if the type of the next DNN data is non-zero.

V. Evaluation Methodology

We use three modern quantized DNNs as benchmarks with different quantization techniques to cover various DNN environments. Table II provides detailed information about the three DNN benchmarks. The quantization datatype is unified to INT8 to make a single word of the same size. We retrain benchmark DNNs with quantization until the difference in accuracy before

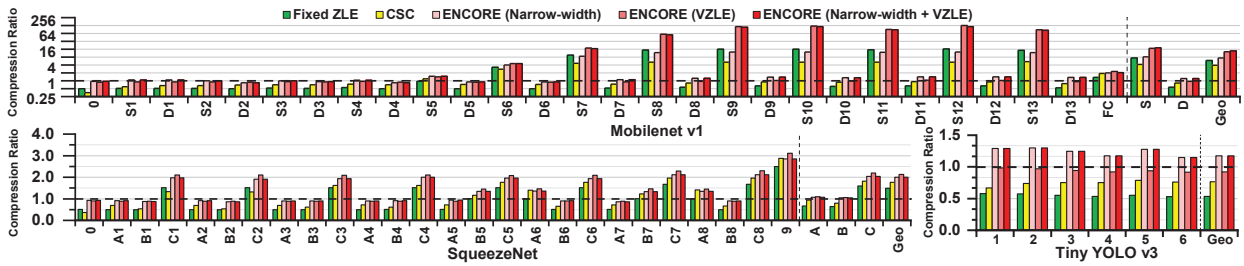

Fig. 6. Weights Compression Ratio of Three DNNs

TABLE II
DNN BENCHMARKS WITH QUANTIZATION TECHNIQUES

DNN	Dataset	Framework	Structure	Quant.
Tiny YOLO v3	COCO	Darknet	LReLU, YOLO	Linear
Mobilenet v1	Cifar10	Tensorflow	Separable Conv.	Min-Max [19]
SqueezeNet	ImageNet	Caffe	Fire Module	Trained [4]

*All three DNNs used the INT8 datatype for quantization.

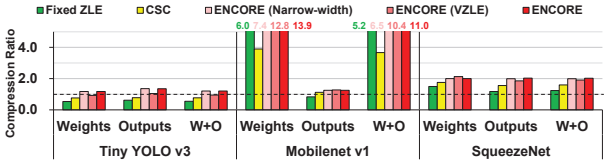

Fig. 7. Compression Ratio

and after quantization is no grater than 5% [3]. For partial output experiments, 10K images are randomly selected from each dataset. The experimental results for the partial outputs are the geometric mean for each image.

We modify the SCALE-Sim [20] simulator to support compression algorithms and ENCORE for evaluating an effective DRAM bandwidth. For the simulation, the DRAM data size is set to 64 bits, followed by Eyeriss v1 [1]. We implement ENCORE in Verilog and measure the area and power consumption using the Synopsys Design Compiler with the TSMC library at 45nm technology. Since Eyeriss v1 and EIE [9], which use the competitive DNN compression algorithms, do not discuss detailed hardware overheads for only the compression engines, we compare hardware overheads of ENCORE to the overall NPU overheads described in papers.

VI. EXPERIMENTAL RESULTS

A. Compression Ratio

We analyze the compression ratio for DNN weights per layer, and Figure 6 shows the results. We separate ENCORE into two parts: the narrow-width value compression and VZLE, and confirm the effect of the compression ratio. The rightmost result for each graph is the average compression ratio through all layers. The previous compression algorithms face difficulty in compressing low sparsity layers such as depthwise layer in Mobilenet ('D' in Figure 6) and squeeze and 1 expand layers in SqueezeNet ('A' and 'B' in Figure 6). However, ENCORE shows a steady compression ratio for every layer. ENCORE suffers compression ratio degradation on SqueezeNet; however, it is not lower than 0.85, whereas other previous compression algorithms fall below 0.5. ENCORE also shows stable compression ratio, 1.17 on average for Tiny YOLO v3. For high sparsity layers such as pointwise layer in Mobilenet v1 ('S' in Figure 6), VZLE shows remarkable efficiency with at most 116 compression ratio while the fixed ZLE faces run-length overflows and obtains a compression ratio lower than 16.

Figure 7 shows the overall compression ratios for the three benchmark DNNs. For all three DNNs, ENCORE shows an average compression ratio of 2.99. The previous compression algorithms fail to compress for Tiny YOLO v3, which has a large narrow-width value distribution. The narrow-width value compression in ENCORE shows a better compression ratio, 1.20 and 2.00 for Tiny YOLO v3 and SqueezeNet, respectively. VZLE shows 10.40 compression ratio for Mobilenet v1. The compression ratio results prove that each compression scheme works appropriately for each DNN with different sparsities.

B. Hardware Overheads of ENCORE

We analyze hardware overheads by synthesizing ENCORE. In the simulation, the ENCORE compressor and decompressor take 8.72 and 3.38 ns per word, respectively. Because iterative type checking is needed to reset the variable zero run-length bits through VZLE, the compressor requires more time than the decompressor. However, the time delay for the compressor does not affect the overall performance because compression is not a critical path. Compression is conducted only when DNN data are evicted from the on-chip buffer. The decompressor does not wait the next data because it resets zero run-length bits by checking whether the read run-length bits are all 1s. Therefore, its latency is less than half of the compressor. Referred by RRAM-DNN [15], decompression latency is short enough, thus the decompressor can decompress DNN data before the PEs require the next inputs or weights, without affecting throughput of PEs computation. EIE [9] takes 12.20 μs to process one frame with 64 PEs. For 64 PEs, the decompressor should decompress 128 words; 64 for inputs and weights each. Because 128 words decompression can be done in 0.43 μs, decompression incurs only 3.52% latency overhead compared to PEs computation time. If we use two ENCORE decompressors for each of weights and partial outputs, latency overhead can be much smaller. Moreover, because ENCORE can be pipelined with DRAM access and PEs computation, its latency cannot be a performance bottleneck.

For area and power consumption, we compare the ENCORE with the previous NPUs. Table III lists the hardware overhead of the three NPUs and ENCORE. Compared to EIE, the ENCORE compressor incurs a 0.02% area and 0.08% power overhead. The ENCORE decompressor incurs a 0.03% area and 0.15%

TABLE III
COMPARISON OF NPUS AND ENCORE

Name	Eyeriss v1 [1]	Eyeriss v2 [2]	EIE [9]	ENCORE	
				Comp.	Decomp.
Tech. (nm)	65	65	45	**45**	
Quant. (bits)	16	8	4	**8**	
Area (mm^2)	16	36.67*	40.8	**0.01**	**0.01**
Power (mW)	278	574	590	**0.48**	**0.89**

*The Eyeriss v2 area is computed based on the Eyeriss v1.

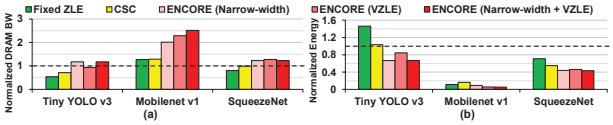

Fig. 8. Normalized (a) DRAM Bandwidth, (b) DRAM Energy Consumption

power overhead. Because the ENCORE overhead is measured as an 8-bit quantization support, these overheads can be smaller if ENCORE is implemented as 4-bit quantization like EIE. Compared to Eyeriss v2 [2], ENCORE incurs a 0.05% area and 0.24% power overhead without considering different technologies. Compared to three well-known NPUs, ENCORE overhead is nearly 1%. The results prove that conventional NPUs can adopt ENCORE with a negligible hardware overhead.

C. DRAM Bandwidth and Energy Consumption

We simulate the DNN inference, and Figure 8 shows the results about effective DRAM bandwidth and energy consumption. The results are normalized to the baseline, which is the case without compression. We assume that three compression algorithms are located between the external DRAM and the on-chip buffer, and focus on the effective DRAM bandwidth which is affected by the three compression algorithms. We simulate the external DRAM to be DDR3 [7].

For the three DNNs, ENCORE shows the highest effective DRAM bandwidth, 1.54 average. For Tiny YOLO v3, the previous compression algorithms suffer low DRAM bandwidth less than 1 with compression failure, but ENCORE suitably compresses DNN data and helps improve effective DRAM bandwidth by 1.17. ENCORE achieves 2.29 effective DRAM bandwidth on Mobilenet v1. This means ENCORE handles twice data at one DRAM access with a small metadata overhead. We also compute the DRAM energy consumption of ENCORE. For Tiny YOLO v3, Fixed ZLE has 1.46 times more DRAM energy consumption compared to the baseline. In contrast, ENCORE consumes 0.67 times DRAM energy by reducing the number of DRAM accesses. The average DRAM energy consumption of ENCORE is 0.25 times of the baseline.

VII. CONCLUSIONS

Modern DNNs do not show high sparsity; thus, previous zero value-aware compression algorithms can not effectively compress DNN data. We observe that quantized DNNs have many narrow-width values. Consequently, we propose ENCORE, which supports narrow-width value compression and VZLE. With three modern DNNs, ENCORE shows an average compression ratio of 2.99 while the other previous compression algorithms suffer to compress. ENCORE incurs 0.05% area and 0.23% power consumption overhead in comparison to EIE.

ACKNOWLEDGMENT

This work was supported by Samsung Electronics Co., LTD (IO201210-07993-01).

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[2] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[3] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1474–1479.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[5] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[6] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th ISCA*. IEEE, 2018, pp. 688–698.

[7] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay, "Adaptive weight compression for memory-efficient neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 199–204.

[8] H.-H. Chin, R.-S. Tsay, and H.-I. Wu, "A high-performance adaptive quantization approach for edge cnn applications," *arXiv preprint arXiv:2107.08382*, 2021.

[9] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[10] S. Kim, J. Kim, J. Lee, and S. Hong, "Residue cache: A low-energy low-area l2 cache architecture via compression and partial hits," in *2011 44th Annual IEEE/ACM International Symposium on MICRO*. IEEE, 2011, pp. 420–429.

[11] J. Hong, H. Kim, and S. Kim, "Ear: Ecc-aided refresh reduction through 2-d zero compression," in *Proceedings of the 27th International Conference on PACT*, 2018, pp. 1–11.

[12] S. Hong and S. Kim, "Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the alu," in *2010 IEEE International Conference on Computer Design*. IEEE, 2010, pp. 342–349.

[13] J. Kim, S. Hong, J. Hong, and S. Kim, "Cid: Co-architecting instruction cache and decompression system for embedded systems," *IEEE Transactions on Computers*, vol. 70, no. 7, pp. 1132–1145, 2020.

[14] J. Kim, M. Kang, J. Hong, and S. Kim, "Exploiting inter-block entropy to enhance the compressibility of blocks with diverse data," in *IEEE International Symposium on HPCA*. IEEE, 2022.

[15] Z. Li, Z. Wang, L. Xu, Q. Dong, B. Liu, C.-I. Su, W.-T. Chu, G. Tsou, Y.-D. Chih, T.-Y. J. Chang *et al.*, "Rram-dnn: An rram and model-compression empowered all-weights-on-chip dnn accelerator," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1105–1115, 2020.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[18] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[19] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for mobilenets," in *2018 1st Workshop on EMC2*. IEEE, 2018, pp. 14–18.

[20] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.