

SqueezeBlock: A Transparent Weight Compression Scheme for Deep Neural Networks

Mo Song, Jiajun Wu, Yuhao Ding, Hayden Kwok-Hay So

Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong

{songmo, jjwu, yhding, hso}@eee.hku.hk

Abstract—Modern Deep Neural Networks (DNNs) are notorious for their large memory footprint, which impacts not only the storage capacity requirement in resource-constrained embedded systems, but also the performance of an inference machine due to data movement. In this work, we demonstrate a transparent weight compression scheme, called SqueezeBlock, which effectively reduces the memory footprint of DNN models with only minimal impact on their accuracy without the need for retraining. SqueezeBlock employs three steps, namely, clustering, quantization, and block encoding, to compress weights of DNN models and relies on automatic design space exploration to derive the optimal encoding configuration. Custom hardware decoders can be generated automatically for seamless integration with the memory subsystem. Experiments on a range of DNNs show that SqueezeBlock can effectively compress the original fp32 weights by up to $4.88\times$ to 6 bit per weight with the loss of accuracy kept within 0.92 % across tested models.

Keywords—Deep Neural Network, Weight Compression, FPGA, Number System

I. INTRODUCTION

The ever-increasing sizes of contemporary deep neural networks (DNN) models have been a major challenge for their deployments in edge scenarios where memory storage is limited and performance is often bottlenecked by data transfer latency. The de facto practice today is to employ model compression techniques such as pruning, quantization, and distillation to reduce the memory footprint of large DNN models before they are deployed [1]. However, to achieve good model accuracy, the majority of existing model compression schemes require extensive retraining of the original DNN with techniques such as quantization-aware training. While such retraining is possible for smaller DNNs, as the model size increases, this retraining process quickly becomes impractical not only due to the long training time, but also due to constraints in training data and computing resources availability, as well as data privacy issues [2], [3].

To address these challenges, we propose SqueezeBlock, a transparent weight compression scheme that automatically compresses weights from a DNN model with a model-specific encoding configuration while maintaining negligible accuracy loss without retraining. SqueezeBlock is an offline weight compression framework with three parts that can be used either

individually or in tandem, namely clustering, quantization, and block encoding. In the full compression scheme with the three steps, the weights are first partitioned into k clusters ($k = 256$ in our current implementation) in the clustering step based on the k -means algorithm. Subsequently, in the quantization step, floating-point quantization of the k centroids is performed using a custom fp8 encoding tailored to the model to minimize quantization error. The weights within a cluster are then assigned to their corresponding quantized centroid value using the same custom fp8 encoding. Finally, in the encoding step, these quantized weight values are grouped into spatial blocks and encoded using a block-based encoding scheme similar to that of block-minifloat. To decode these weight values during run time, the framework can automatically generate a hardware decoder for integration with the memory subsystem, making the weight compression process entirely transparent to the inference machine.

Depending on the targetted accuracy-compression trade-off, some of the steps may optionally be omitted. For example, quantized floating point weights such as fp8 can be obtained directly without using the block encoding step. A design space exploration (DSE) scheme is presented that can help determine the steps and encoding that are necessary to achieve minimal accuracy drop for a given DNN and desired weight compression ratio.

SqueezeBlock has been tested on a range of vision DNNs and BERT with a compressed bitwidth of up to 5 bits per weight. The results show that SqueezeBlock results in favorable tradeoff in compression versus accuracy drop. With block-based 6-bit SB6 encoding, all tested models achieved accuracy within 0.92 % of the original fp32 models while achieving $4.88\times$ memory compression ratio. With non-block-based 8 bit FP8 encoding, less than 0.62 % drop in accuracy was achieved across all models tested. The key contributions of this work are thus in:

- We propose SqueezeBlock, a memory-friendly flexible block-based weights compression scheme without the need for retraining.
- We design an automatic end-to-end DSE framework to search for the optimal compression configurations for any given DNN model.
- A hardware generator is developed to automatically produce a lightweight hardware decoder based on the DSE results.

This work was supported in part by the Research Grants Council (RGC) of Hong Kong under the Research Impact Fund project R7003-21 and in part by the AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Hong Kong SAR.

II. BACKGROUND & RELATED WORKS

A. DNN Models Compression

Compression techniques have been proposed to address the issues of deploying large-scale DNN models on edge devices / IoT while preserving their accuracy. Various works have been proposed discussing weight pruning to reduce storage requirements and achieve fast inference [4]. Quantization aims to reduce the data precision & bitwidth to squeeze the storage consumption and computation cost [5]. Low-rank factorization focuses on approximately compressing the matrix, thus reducing memory cost, with the requirement of reliable approximation methods for different models and layer types [6]. In addition, knowledge distillation is used to fine-tune a large and cumbersome model and deploy it into a smaller and lighter network [7]. However, most of these methods require retraining of compressed models to maintain performance. In our work, we focus on the weight compression scheme with an evaluation of the encoding configuration based on a cluster-aided floating-point quantization method without retraining overhead.

B. Weight Clustering

Clustering, or weight sharing, reduces the number of unique weights in a model, leading to benefits for compression. The weights of each layer are first categorized into clusters, and all the weights belonging to the cluster share the same centroid value. For instance, GOBO [8] employs an outlier-aware quantization technique that distinguishes between outliers and Gaussian distributed weights by retaining outliers in their uncompressed form while clustering the remaining weights into eight representative values, accessible via 3-bit indexes. This approach, although not fine-tuning required, is specifically designed for the BERT model [9] and is not directly applicable to general CNNs, which typically need at least $2^7 = 128$ clusters to mitigate performance loss [10]. In this work, we start from the K-Means algorithm and make a variant to better support our compression scenario.

C. Low-bitwidth Floating-point Quantization

Quantization is a widely used technique in DNN models to improve memory and computational efficiency. Although integer quantization is the conventional approach, low-bitwidth floating-point schemes (such as FP8) have been shown to offer better generality and accuracy preservation due to their nonlinear mapping [11]. A low-bitwidth floating-point (LFP) number can be represented by a set of elements as Eqn. (1):

$$LFP = \begin{cases} (-1)^s \times 2^{exp-bias} \times (1 + man \times 2^{-M}) & exp > 0 \\ (-1)^s \times 2^{1-bias} \times (man \times 2^{-M}) & exp = 0 \end{cases} \quad (1)$$

where $s \in 0,1$ is the sign bit, the exponent (exp) and the mantissa (man) fields are unsigned data of the E bit and the M bit, respectively. We use the notation $ExpMy$ to denote an LFP format with an x -bit exponent and a y -bit mantissa. In general cases, the exponent bias ($bias$) is equal

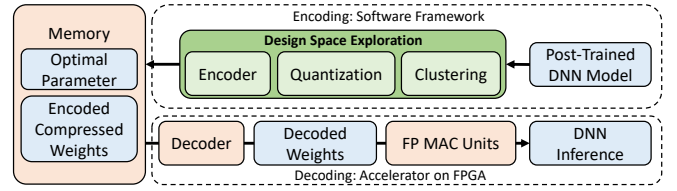


Fig. 1: SqueezeBlock Framework Overview

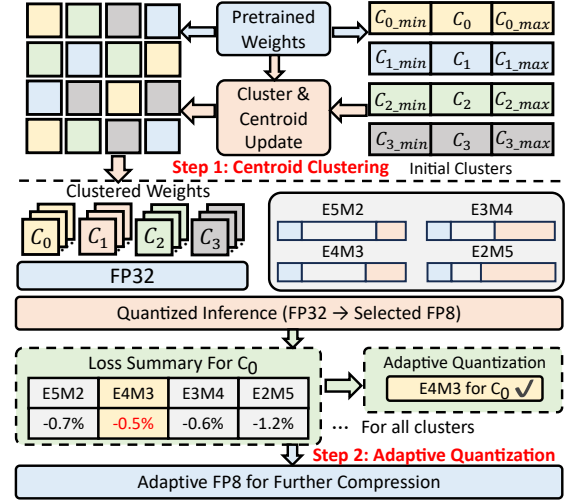


Fig. 2: Weight Clustering and Adaptive FP8 Quantization Step of SqueezeBlock

to $2^{E-1} - 1$. Standard floating-point numbers have normalized ($exp > 0$) and subnormalized ($exp = 0$) formats. We ignore the subnormalized format in the following discussion and assume that the representation is the same as the normalized one when $exp = 0$. When the total length of the LFP is fixed as $((1 + E + M)\text{-bit})$, we can change the exponent and the length of the mantissa field to get different number grids [12]. Therefore, the dynamic range can be maximized in the granularity of each layer of the neural network by dynamically adapting its exponent range and optimally removing its representable data points [13].

III. SQUEEZEBLOCK FRAMEWORK

SqueezeBlock compresses DNN weights using a block-based method and generates a hardware decoder that decodes compressed weights based on different parameter settings. Fig. 1 presents the overview of the proposed framework, where end-to-end weight compression is performed based on an encoding and a decoding scheme. The objective of compression is to make the encoded format represent as much data information as possible from the original ones. The compression process involves three primary stages: weight clustering, FP8 quantization, and DSE-optimized encoding. This section provides a thorough explanation of these steps and also covers the hardware decoder.

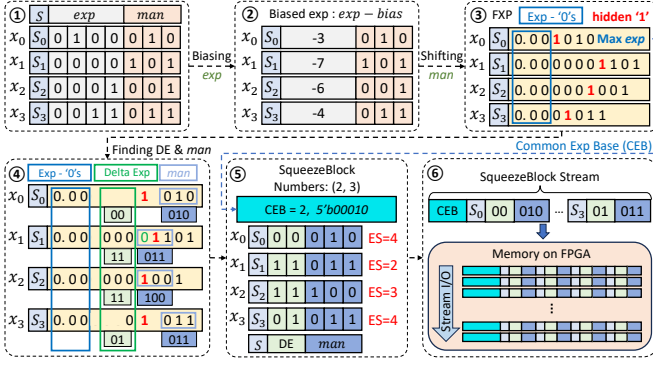


Fig. 3: SqueezeBlock Encoder. The example presented in this figure is a four-entry vector ($x_0 \sim x_3$). The dash boxes refer to different substeps with a special case in the proposed encoding scheme.

Algorithm 1 Weight Clustering Method

Input: Post-trained DNN weights $\mathbf{W}$, maximum iteration T
Output: Clustered centroids $\mathbf{C} = (c_1, c_2, \dots, c_k)$

- 1: Linearly initialize centroids $\mathbf{C}_0$
- 2: $D \leftarrow$ Absolute distance $\sum_{i=1}^k \sum_{w \in c_i} |w - c_i|$.
- 3: **while** Iteration $t \leq T$ or D is not optimal **do**
- 4: Update $\mathbf{W}_t = (w_1, w_2, \dots, w_n)$ to the nearest cluster
- 5: Compute the mean of clusters to update $\mathbf{C}_t$
- 6: $D \leftarrow$ Update absolute distance $\sum_{i=1}^k \sum_{w \in c_i} |w - c_i|$.
- 7: **end while**

A. Encoding

1) *Centroid Clustering*: Conventional K-Means algorithm uses random initialization which can lead to poor representation when outliers are present in the weight distribution. Instead, we use a linear initialization for cluster centroids, with higher resolution in populated weight areas, as illustrated in Algorithm 1. As shown in Fig. 2 (Step 1), we sort and categorize the weights into each cluster by the closest absolute distance and use the average weights within the cluster as its centroid. The sum of absolute distances represents the total variation within the clusters. New centroids are updated as means in each cluster, and this process continues until the total variation is minimized.

We found that clustering requires on average 256 centroids for retaining the accuracy models without fine-tuning, and larger numbers of centroids generally result in higher performance. As our compression scheme is general and robust with various encoding formats, the outlier problem that exists in some conventional clustering methods can be solved by the proposed linear initialization and a flexible dynamic range of floating-point representations.

2) *Adaptive FP8 Quantization*: To reduce the truncation error of range and precision, the clustered weights are further quantized from FP32 to FP8 as shown in Fig. 2. The choice of each bitwidth for mantissa and exponent would affect the trade-off between precision and dynamic range. In this work, we set four FP8 candidates with the formats E5M2, E3M4,

E4M3, and E2M5. Each centroid is processed by all candidates for the FP8 quantizer to estimate the loss compared to the full-precision baseline (FP32) and find the optimal format with the lowest loss.

By applying the quantization step, two primary advantages are achieved: First, the distribution of centroids values may not fit a fixed FP8 format. Therefore, the optimal FP8 format for each centroid can be achieved by adaptively assigning different FP8 formats to the weights. Second, the traditional mixed-precision method usually has an exponentially huge computation overhead to find the optimal selection. However, in our case: 1) we only set up four reasonable FP8 formats as the candidates to reduce the search space, and 2) by clustering, there is only on average 256 FP32 centroids, which is much fewer than the number of weights in a DNN layer, scaling down the search space further.

3) *Block-based Compression Scheme*: After the adaptive FP8 quantization, all the weights should be encoded for compressed storage. SqueezeBlock uses a block-based encoding for a set of weights as a block. Each encoded weight x_i of an element- i of a block floating-point tensor x with a sign s_i , a m -bit fixed point mantissa man_i , an e -bit delta exponent (DE) exp_i , and a b -bit common exponent base (CEB) exp_x of the block, its real value can be computed as:

$$x_i = (-1)^{s_i} \times man_i \times 2^{-(exp_i + exp_x)} \quad (2)$$

Note that all exponent values are previously biased in the following discussion (i.e., all values of exp have been subtracted by $bias$). By specifying different bitwidths m and e , the optimal encoding configuration can be evaluated and used to compress elements within the block. Subsequently, for a floating-point tensor block x with N elements, the encoded block can be stored in memory with encoding fields efficiently.

Fig. 3 detailedly illustrates how to encode a block of FP8 weights x quantized by E4M3 (①) into 5-bit CEB and 6-bit encoded tensors with a particular encoding setting (1-bit sign s_i , 3-bit mantissa man_i and 2-bit delta exponent exp_i). As shown in ②, each exponent is used to convert the weight to the fixed-point format in ③ by shifting mantissa (1.XXX). In this step, the position of hidden bits (highlighted in red) of all weights can be determined, and the CEB of this block can be found by subtracting one from the maximum biased exponent in ②. In this case, the exponent of x_0 is the CEB ($exp_x = 2$, $5'b00010$) for the block.

The next step ④ is to determine the representative delta exponent (DE) and mantissa man_i for each weight x_i . As covered by the green frame, since there is no zero before the leading '1', x_0 has a zero DE ($exp_0 = 2'b00$). Therefore, the mantissa field can store all 3-bit values from bits after the leading '1' ($man_0 = 3'b010$), in which the leading '1' has been hidden again. In the same way, x_1 has four '0's ahead of the leading '1', forcing the leftmost '0' to occupy the mantissa field (the green '0') since the DE is only 2-bit. Therefore, the DE of x_1 is 3, and move the leftmost '0' to occupy one position in the mantissa field. As a result, the

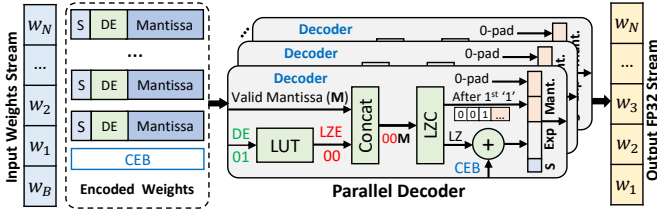


Fig. 4: Hardware decoder for the proposed SqueezeBlock

leading ‘1’ cannot be hidden in $man_1 = 3'b011$. Similarly, exp_2 of x_2 is 3, and there is no ‘0’ flushed into the mantissa field, maintaining higher precision after encoding. For x_3 , since there is only one ‘0’ before the leading ‘1’, we can keep all 3-bit mantissa of FP8 within $man_3 = 3'b011$ and use $exp_3 = 2'b01$ as the DE. After extracting CEB, DE, and mantissa from this block, the encoded weights are shown in ⑤. All the compressed weights are stored in memory, and block-based memory streaming is shown in ⑥.

We use the number of Effective Significands (ES) as a metric to measure the effectiveness of an encoding parameter setting. The ES is defined as the count of valid mantissa bits that can be stored from the original FP32 data in the encoded format. In the given example of Fig. 3 ⑤, ES represents the number of mantissa bits retained for a given encoding setting with m -bit man_i , including hidden ‘1’. In the cases of x_0 and x_3 , all three bits in the deep blue box originate from the mantissa field of the FP8 representation, so the ES is calculated as 4 (3 explicit mantissa bits and 1 implicit hidden ‘1’). For x_1 , since the leading ‘1’ cannot be hidden, the implicit ‘1’ and 1-bit mantissa of the MSB sum up to the ES value of 2. Similarly, since the man_2 is occupied with the hidden ‘1’ and two bits from mantissa, this encoded format gives 3 ES.

B. Decoding

To support the SqueezeBlock scheme, we design a hardware architecture with a specific decoder to convert encoded weights in SqueezeNumber into standard floating-point (FP) numbers. The overall architecture consists of the memory interface, weight decoder, data buffers, and a systolic array. Since we only target floating-point-based weight compression, the activations remain original numbers, and all the multiply-accumulation (MAC) operations inside processing elements are based on the FP format. The systolic array can handle convolutional layers (by `im2col` pre-processing), fully-connected layers, and self-attention blocks in almost all the DNN models by matrix multiplication. Fig. 4 shows the weight decoder design. The input stream of weights goes through the decoder and the number of weights in each block is defined by the SqueezeNumber format. Each encoded weight is sent to a decoder to generate the FP32-based exponent, mantissa, and sign bit. Exponents are calculated by the shared base in the block and the DE in each encoded weight. For the mantissa part, it receives the valid mantissa, concatenates it with leading zeros based on the exponent (LZE) and finds the first ‘1’ using a leading zero detector (LZD). All bits following the first ‘1’

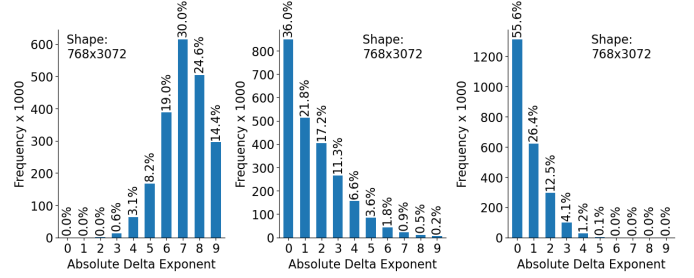


Fig. 5: Histograms of the absolute biased exponent (left), the absolute delta exponent (middle) of FP32 weights, and the absolute delta exponent (right) of FP8 weights on the ViT.

are regarded as the final mantissa, and the decoder expands them to the full 23-bit field in standard IEEE FP32 format by padding ‘0’ bits. The decoded weights are used for FP32-based MAC operations in the following systolic array. The output activations are also stored as floating-point in the on-chip buffer and will finally be written back to the external memory. Since our compression scheme is completely static, the only overhead introduced to hardware is the weight decoder, and our evaluation results illustrate that the overhead is quite small and negligible. The hardware decoder is designed as a Vitis HLS-based template so that it can be automatically generated by the SqueezeBlock framework with the optimal parameters.

IV. EVALUATION

A. Experiments Setup

We evaluated the proposed SqueezeBlock with a series of experiments based on several vision models, VGG-11 & VGG-16 [14], ResNet-18 & ResNet-50 [15], and Vision Transformer (ViT) [16], and a large language model, BERT [9]. The dataset for the vision models is ImageNet-1k [17], and for the BERT model, we evaluate it on MNLI language tasks [18]. Specifically, we evaluated the quantization, DSE and hardware performance and finally gave comparisons with prior work.

B. Cluster-Aided FP8 Quantization Evaluation

Fig. 5 shows histograms with absolute delta exponent distributions of the ViT model. The figure on the left shows the absolute biased exponent distributions for FP32 weights, formulating a more centralized distribution and primarily relying on larger values. By applying a block-based encoding method, the middle figure indicates that our encoding method, utilizing a CEB, can replace raw FP32 absolute biased exponents in the left figure with the absolute DE. Furthermore, if we apply the method after a cluster-aided FP8 quantization, the distribution is more left-skewed (right figure), with DE primarily concentrated on smaller values. For the ViT model, the most absolute biased exponents of FP32 weights on the left of Fig. 5 focus on 6, 7, and 8. Upon applying block-based encoding, most DE settle at values of 0, 1, and 2 with roughly 75% of all DE, which can be more encoding-friendly. Furthermore, the proposed cluster-aided quantization method further consolidates DE to the minimal values of 0,

1, and 2 with 94.5% of the total distribution, reflecting a 19.5% improvement compared to direct compression of the middle row. It can be concluded that the proposed method can significantly help to improve the efficiency and adaptability of encoding due to smaller DE.

C. DSE and Performance Evaluation

Table I presents the top-1 accuracy performance for the selected DNN models based on the optimal (e, m) configurations, where the corresponding compression ratio of each block-based encoding setting is illustrated on the right. We use the total number of bits inside each block (block bits) to illustrate the compression ratio. Assuming that a b -bit CEB is shared for N elements within a block, the block bits can be calculated as $BlockBits = b + N \times x$, where x refers to x -bit SqueezeBlock numbers. Reducing the total number of bits within a block results in an increased compression ratio, which can reduce the memory bandwidth required for transferring weights between memory and the inference accelerator. Thus, there is a trade-off between accuracy and hardware efficiency. For instance, reducing the number of bits from 7-bit to 6-bit SB decreases the block bits from 68 to 59 with a slight reduction in accuracy. (-0.22% in ViT model). 59 bits require 8 bytes of memory storage, which is aligned with the commonly used 64 bytes cache line in modern computer architectures. This alignment can improve memory access efficiency, as each cache line can be loaded or stored in a single memory transaction. Conversely, 68 bits weights require 9 bytes of memory storage, which is not aligned with the 64 bytes cache line. This misalignment can lead to inefficient memory access, as each cache line may only partially contain a block of weights, resulting in additional memory transactions and reduced memory bandwidth.

D. Ablation Studies

We conduct extensive ablation studies on each proposed method. As Table II shows, we evaluate our methods on VGG-16BN, ResNet-18, and ViT-B models. The baseline direct compression scheme experiences a substantial loss in accuracy, particularly when operating under an 8-bit setting. On the contrary, our proposed methods preserve accuracy and exhibit increasingly apparent advantages at lower bitwidths. Although the application of the clustering method provides a slight improvement in accuracy, the use of FP8 quantization significantly enhances performance. As an illustration of the effectiveness of our proposed methods, applying clustering to ViT-B models with 6-bit weight encoding resulted in a 3.76% accuracy improvement, yielding an accuracy of 78.89%. In comparison, direct compression without preprocessing methods yielded an accuracy of 75.13%. Furthermore, the impact of FP8 quantization was found to be more substantial than that of the clustering method, resulting in an accuracy of 80.32% for ViT-B models tested with 6-bit weight encoding, which is 5.19% higher than that achieved through direct compression. Furthermore, the joint application of the proposed methods yields a further performance improvement.

E. Hardware Utilization

We implemented the decoder in HLS with different parameter settings. The post-synthesis resource utilization of a single decoder implemented in Xilinx Ultra96 for our direct compression method is shown in Table III. DSPs and BRAMs are not used in the decoder and thus are omitted in the table. All parameter settings can be implemented within hundreds of LUTs and FFs. Our decoder can be seamlessly embedded into edge devices with minimal hardware overhead.

F. Comparison with State-of-the-art Methods

There have been some works that target the prolific problem of data channel saturation. Specifically without using any retraining, ZHW [19] reduces data throughput bottlenecks in scientific computation by compressing and decompressing the numerical fields and stored to a file system. ZHW uses a block floating point method to represent the 4^d d-dimensional data values in a block using a common exponent to form a binary matrix with 4^d rows that can hold these coefficients. It can be seen that the throughput of ZHW is intrinsically limited since it respects block-level data dependencies between binary matrices. However, since the primary target of ZHW is for scientific data simulation, it is not specific to the neural network attributes, where each layer with a particular dimension to extract features from input images and intrinsic data dependencies are important to keep the information learned from images within each dimension. Our SqueezeBlock compression scheme can support freely-chosen block sizes from low to high granularity and different encoding parameter settings with respect to effectual significands evaluated to guarantee the optimal performance with a much fewer hardware resource utilization compared to ZHW as shown in Table IV.

V. CONCLUSIONS

In this work, we present SqueezeBlock, a transparent weight compression scheme to effectively reduce the memory footprint of DNN models while maintaining a low accuracy degradation without requiring network retraining. SqueezeBlock uses a cluster-aided FP8 quantization method to preprocess weights that facilitates a subsequent block-based weight encoding scheme to achieve a good compression ratio and accuracy loss tradeoff. In addition, the proposed design space exploration framework allowed us to identify optimal encoding parameter configurations for different DNN models. A hardware decoder customized to the specific encoding could then be generated automatically to decode the weights during run time. Experiments showed that SqueezeBlock was able to retain the most significant information from the original model despite the large compression ratio, which may not be immediately apparent, as demonstrated by the ViT model compared to other vision models. In the future, we plan to expand the proposed scheme to compress and decompress activations during run time on hardware, which can improve both DNN inference and training time.

TABLE I: Performance of the compression scheme with various encoding bitwidths evaluated with the top-1 accuracy(%) on the ImageNet and MNLI tasks. The (e,m) configuration is optimally searched based on ES. The compression ratio is the scale of block bits compressed for each data type with respect to the FP32 baseline. Block size is 9.

Encoding	VGG-11	VGG-16	ResNet-18	ResNet-50	ViT-B	BERT	Block Bits	Compression Ratio
FP32 (baseline)	82.16	82.42	74.13	79.94	81.79	84.42	288	1.00×
FP8 $(E,M)^*$	82.03 (4,3)	82.21 (4,3)	73.90 (4,3)	79.32 (4,3)	81.46 (5,2)	84.15 (3,4)	72	4.00×
SB7 $(e,m)^\dagger$	81.86 (2,4)	81.88 (2,4)	73.61 (2,4)	79.24 (2,4)	81.35 (1,5)	83.97 (1,5)	68	4.24×
SB6 $(e,m)^*$	81.67 (2,3)	81.65 (2,3)	73.21 (2,3)	79.03 (2,3)	81.27 (1,4)	83.65 (1,4)	59	4.88×
SB5 (e,m)	76.26 (2,2)	77.28 (2,2)	68.82 (2,2)	74.91 (2,2)	79.80 (1,3)	80.83 (1,3)	50	5.76×

* (E,M) : FP8 with E -bit exponent and M -bit mantissa. † SB x : x -bit SqueezeBlock. * (e,m) : e -bit DE with m -bit mantissa.

TABLE II: Effect of clustering method and FP8 quantization on compression scheme. We evaluate the accuracy performance (%) of all models on the ImageNet.

Encoding	Cluster	FP8 Quant	VGG-16	ResNet-18	ViT-B
SB7 † $(e,m)^*$	$\times$	$\times$	81.03 (2,4)	72.78 (2,4)	80.76 (1,5)
	$\checkmark$	$\times$	81.26 (2,4)	73.02 (2,4)	80.93 (1,5)
	$\times$	$\checkmark$	81.62 (2,4)	73.27 (2,4)	81.21 (1,5)
	$\checkmark$	$\checkmark$	81.88 (2,4)	73.61 (2,4)	81.35 (1,5)
SB6 (e,m)	$\times$	$\times$	76.94 (2,3)	68.26 (2,3)	75.13 (1,4)
	$\checkmark$	$\times$	78.37 (2,3)	70.49 (2,3)	78.89 (1,4)
	$\times$	$\checkmark$	80.84 (2,3)	72.44 (2,3)	80.32 (1,4)
	$\checkmark$	$\checkmark$	81.65 (2,3)	73.21 (2,3)	81.27 (1,4)
SB5 (e,m)	$\times$	$\times$	70.36 (2,2)	58.23 (2,2)	71.70 (1,3)
	$\checkmark$	$\times$	72.94 (2,2)	60.53 (2,2)	74.25 (1,3)
	$\times$	$\checkmark$	75.36 (2,2)	66.43 (2,2)	77.47 (1,3)
	$\checkmark$	$\checkmark$	77.28 (2,2)	68.82 (2,2)	79.80 (1,3)

† SB x : x -bit SqueezeBlock. * (e,m) : e -bit DE with m -bit mantissa.

TABLE III: Resource Utilization of SqueezeBlock Decoder

$(e,m)^*$	(6,1)	(1,6)	(5,2)	(2,5)	(4,3)	(3,4)
LUT	0.50%	0.46%	0.55%	0.78%	0.48%	0.63%
FF	0.17%	0.17%	0.21%	0.22%	0.17%	0.17%

* (e,m) : e -bit DE with m -bit mantissa.

REFERENCES

- [1] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A Comprehensive Survey on Model Compression and Acceleration," *Artificial Intelligence Review*, vol. 53, pp. 5113–5155, 2020.
- [2] Z. Huo, Z. Zhou, and X. Zhuang, "Are Quantized Models Viable for Large Scale Applications?" in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 13 328–13 337.
- [3] H. Bai, L. Hou, L. Shang, X. Jiang, I. King, and M. R. Lyu, "Towards Efficient Post-training Quantization of Pre-trained Language Models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1405–1418, 2022.
- [4] Y. Gao, B. Zhang, X. Qi, and H. K.-H. So, "DPACS: Hardware Accelerated Dynamic Neural Network Pruning through Algorithm-Architecture Co-design," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 237–251.
- [5] J. Wu, J. Zhou, Y. Gao, Y. Ding, N. Wong, and H. K.-H. So, "MSD: Mixing Signed Digit Representations for Hardware-efficient DNN Acceleration on FPGA with Heterogeneous Resources," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2023, pp. 94–104.
- [6] C. Li and C. Shi, "Constrained Optimization Based Low-Rank Approximation of Neep Neural Networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 732–747.

TABLE IV: Comparison of hardware utilization and accuracy performance (%): SqueezeBlock vs ZHW.

	LUT	FF	Granularity	VGG-16	ResNet-18
SB8 (5,2) †	390	299	High	82.3 (-0.1)	74.0 (-0.2)
ZHW	14324	19974	Low	81.6 (-0.8)	73.7 (-0.5)

† SB x : x -bit SqueezeBlock. (5,2) refers to 5-bit DE with 2-bit mantissa.

- [7] X. Zhu, S. Gong *et al.*, "Knowledge Distillation by On-the-Fly Native Ensemble," *Advances in neural information processing systems*, vol. 31, 2018.
- [8] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.
- [9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [10] S. Son, S. Nah, and K. M. Lee, "Clustering convolutional kernels to compress deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 216–232.
- [11] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort, "FP8 Quantization: The Power of The Exponent," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 651–14 662, 2022.
- [12] X. Sun, J. Choi, and et al., "Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks," *Advances in neural information processing systems*, vol. 32, 2019.
- [13] T. Tambe, E.-Y. Yang, and et al., "Algorithm-Hardware Co-design of Adaptive Floating-point Encodings for Resilient Deep Learning Inference," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.
- [15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '16. IEEE, Jun. 2016, pp. 770–778.
- [16] A. Dosovitskiy, L. Beyer, and et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2021.
- [17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A Large-Scale Hierarchical Image Database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [18] A. Williams, N. Nangia, and S. R. Bowman, "A Broad-Coverage Challenge Corpus for Sentence Understanding Through Inference," 2017.
- [19] M. Barrow, Z. Wu, S. Lloyd, M. Gokhale, H. Patel, and P. Lindstrom, "ZHW: A Numerical CODEC for Big Data Scientific Computation," in *2022 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2022, pp. 1–9.