Adaptive Weight Compression for Memory-Efficient Neural Networks

Jong Hwan Ko, Duckhwan Kim, Taesik Na, Jaeha Kung, and Saibal Mukhopadhyay
School of ECE, Georgia Institute of Technology, Atlanta, Georgia, USA
{jonghwan.ko, kimduckhwan, taesik.na, jhkung}@gatech.edu, saibal@ece.gatech.edu

***Abstract*— Neural networks generally require significant memory capacity/bandwidth to store/access a large number of synaptic weights. This paper presents an application of JPEG image encoding to compress the weights by exploiting the spatial locality and smoothness of the weight matrix. To minimize the loss of accuracy due to JPEG encoding, we propose to adaptively control the quantization factor of the JPEG algorithm depending on the error-sensitivity (gradient) of each weight. With the adaptive compression technique, the weight blocks with higher sensitivity are compressed less for higher accuracy. The adaptive compression reduces memory requirement, which in turn results in higher performance and lower energy of neural network hardware. The simulation for inference hardware for multilayer perceptron with the MNIST dataset shows up to 42X compression with less than 1% loss of recognition accuracy, resulting in 3X higher effective memory bandwidth and ~19X lower system energy.**

Keywords—neural network; weight; compression; memory-efficient; JPEG; MLP

I. INTRODUCTION

In recent years, neural networks have achieved great successes in many computer vision applications including image classification [1], pattern recognition [2], and gesture detection [3]. Generally, complex neural networks contain a large number of synaptic "weights", and the size of the weights increases with the number of layers. The increasing number of weights translates to more demand on memory capacity as well as memory bandwidth. The memory demand is a key challenge for neural computation, especially for resource-constrained platforms such as mobile systems [4]. Figure 1 shows the memory demand as a function of image size for a multi-layer-perceptron (MLP) based neural network for handwritten digit classification with the MNIST dataset [5]. It is evident that even for a moderate image size, an on-chip memory of a mobile processor is not sufficient to store all the weights, so the weights should be stored in an external memory. In such a design, system performance and energy are strongly determined by the performance and energy associated with moving data between memory and computation logic. The recent trend in processors show that energy for accessing data from external memory (70 pJ/bit for DDR3 [6]) is far larger than that of on-chip memory access (0.16 pJ/bit for 28nm SRAM [7]) or computation (16 pJ/16-bit multiply-accumulate operation with 28nm process [8]). Consequently, a neural network architecture that requires storing and/or accessing a large number of weights in an off-chip memory degrades the energy-efficiency of the system.

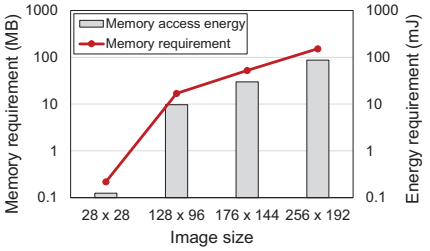

Figure 1. Required memory and energy with different image size using MLP-based neural network for MNIST dataset.

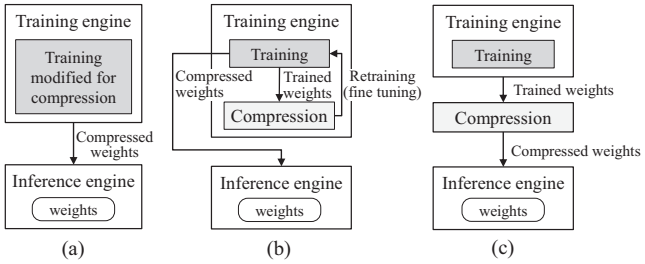

Figure 2. Three different weight compression schemes. (a) Compression requiring modified training algorithm, (b) Compression requiring fine tuning, and (c) proposed training-independent compression scheme

One approach to address the memory energy and performance challenge in neural networks is the processing-in-memory architecture, as presented by Kim et al. [7]. However, such approaches require innovation/development of new memory and integration (packaging) technology. The orthogonal and complementary approach is to reduce the storage requirement by compressing the weights while preserving the accuracy of the network [10]-[15] (see section II for details). However, most of these approaches require modification of training process [9] or retraining (fine-tuning) [4] to minimize the accuracy loss [Figure 2(a), (b)].

This paper presents an approach to compress the weights of an already-trained network without modifying the baseline training process and/or using re-training [Figure 2(c)]. Inspired by smoothness and spatial locality of the weights, we propose to apply an image compression technique on the weight matrix (instead of applying data encoding techniques such as pruning on the individual weight). The key contributions of this paper are:

- We propose to apply JPEG encoding to compress the entire weight matrix as a single image. To our knowledge, this is the first work to apply an adaptive image encoding algorithm for compressing the weights in neural networks.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

- For higher compression with minimum loss of accuracy, we propose to adaptively control the JPEG encoding parameter (quality factor) depending on the error-sensitivity (gradient) of the weights, which is available from the training process.
- We present the design of an inference engine with a JPEG decoder embedded in the memory controller. We study the impact of weight compression on the memory requirement, performance, and energy consumption of a hardware system for inference, by considering memory access energy/latency and decoder overhead.

The compression performance of the proposed approach is demonstrated with a multilayer perceptron (MLP) based neural network for three benchmark datasets. The proposed method achieves 42X compression at the expense of ~1% degradation in recognition accuracy for MLP with the MNIST dataset. For system-level performance and energy analysis, a digital neural network inference engine with an JPEG decoder embedded in the memory controller is synthesized in 28nm CMOS technology. The proposed approach shows 3X higher effective memory bandwidth and ~19X lower system energy for inference.

II. Related Work

There have been various approaches to compress the weights of a neural network to enhance the memory-efficiency. One simple way is to reduce the bit-precision of the weights to reduce both computing energy and storage requirements as demonstrated in [10]. However, the compression is limited by the lowest bit precision, for example, reducing precision from 32 bit to 4 bit will only result in 8X compression. To achieve higher compression, approaches have been explored to reduce the number of weights, for example, by neglecting weights (pruning connections) with magnitude below a threshold [11]. Another approach is the low-rank matrix approximation, which uses factorization to reduce the representation by two smaller matrices [12]. For lossless compression of output images of convolutional neural networks, Chen et al. used a run-length-based encoding technique [13]. There are a few works that compress the weights by transforming them into the frequency domain. Koutník et al. has presented a method to discard high frequency components of the weights to meet the compression requirement [14]. Chen et al. has proposed to random hashing of the frequency components of convolutional neural network filters into smaller set of hash buckets [15].

Most of the preceding approaches for reducing the weight size require changing the training process to minimize the accuracy loss. For example, the approach in [4] utilizes fine tuning (retraining) after each compression stage to maintain the original accuracy. Courbariaux et al. has proposed a specialized training algorithm for the binary representation of the weights [9]. If the prior compression approaches are applied without modification of training algorithm (or re-training), the accuracy drop can be significant [16].

The need to modify training process adds complexity in designing inference hardware. In response, we propose an adaptive image encoding technique for weight compression that does not involve the training process. In addition, while the prior works have focused entirely on reducing the memory capacity, we analyze the impact of compression and its time/energy overhead on the system energy and performance.

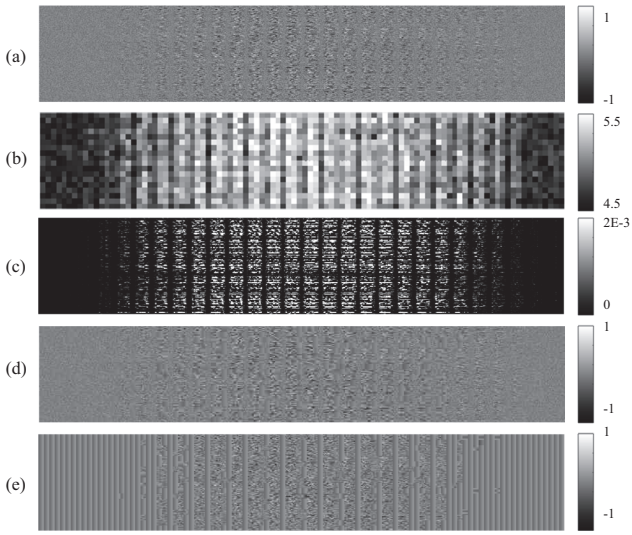

Figure 3. Visualization of a weight matrix of MLP for MNIST. (a) Original weight matrix, (b) average entropy, (c) absolute error sensitivity, (d) compressed matrix using JPEG with single QF, and (e) compressed matrix using JPEG with multi QF.

III. Preliminaries

A. Error Sensitivity of Weights

In the supervised learning of feedforward neural networks, each synaptic weight w_{ji} is updated so that the error in the output layer can be minimized [17]. To update the weight, the backpropagation algorithm computes the sensitivity (gradient) of each weight to the output error ($\partial \mathrm{E}/\partial w_{ji}$) from output layer back to input layer. In recent studies, this information has been utilized to design low-power neural networks through approximate computing [8]. In this work, we utilize the error sensitivity of the weights for its compression, by using an adaptive image encoding concept.

B. JPEG

JPEG [18] is a common encoding method for compressing images. A source image into JPEG is first divided into 8x8 block, and each block is transformed into the frequency domain by discrete cosine transform (DCT). Then, the 64 DCT coefficients are quantized in conjunction with a quantization Table, which is specified by the quality factor (QF). The quantization process is a principal source of losing the original information, and the QF is used to control the lossiness. The final step, entropy coding, achieves additional compression by losslessly encoding the quantized DCT coefficients based on their statistical characteristics.

IV. Adaptive Image Compression of Weights

A. Motivation

The weight parameters in each layer of neural networks can form a 2-D matrix, and can be represented as an image with each pixel being the corresponding weight value. Figure 3(a) shows visualization of a 784 x 144 matrix of weights in the

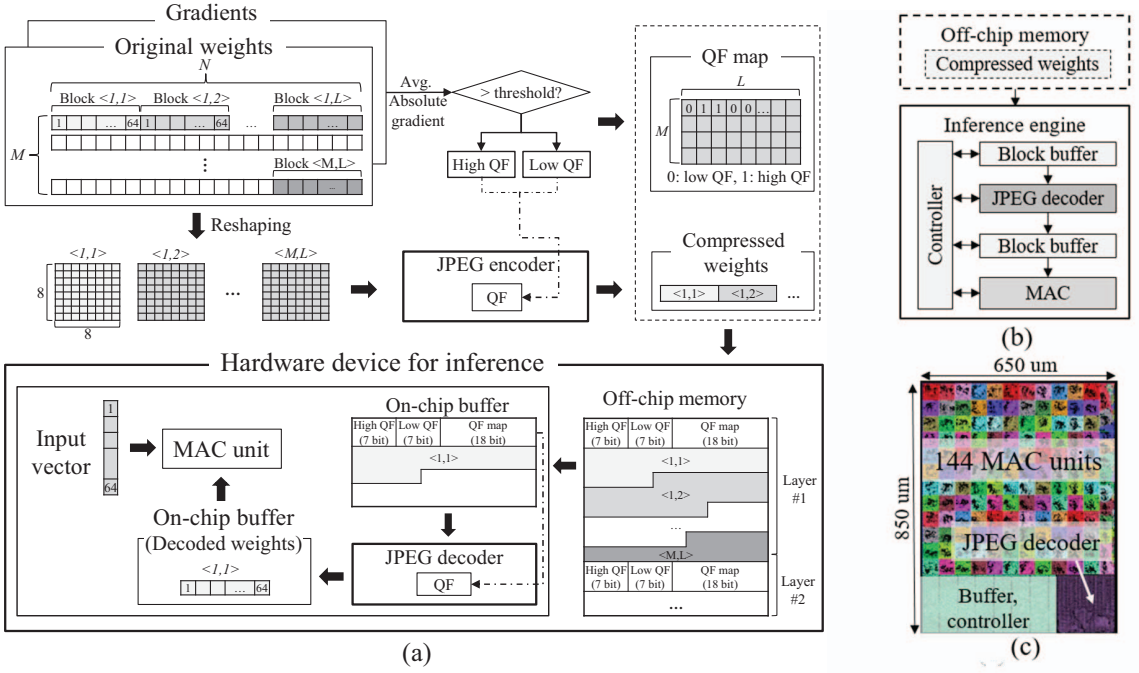

Figure 4. (a) Overall process of the proposed compression approach. (b) Block diagram and (c) layout of the inference hardware.

first fully-connected layer of a MLP network trained for MNIST dataset. As seen in the figure, a large portion of the image is smooth, and has certain types of pattern (redundancy). This is known to happen due to the nature of spatial locality (local pixel correlation) in the input dataset of the network [15]. As images with smoothness can be efficiently compressed by image encoding techniques, we propose to apply JPEG encoding for compressing the weight parameters.

The compression performance achieved by image encoding is related with the entropy theory [19]. The entropy is the statistical measure of the amount of information contained in an image, and used to describe its `business' or 'randomness'. Areas with smoothness or redundancy generally have lower entropy and can be compressed more. Figure 3(b) shows entropy distribution of the sample weight matrix in a unit of an 8x8 block. It can be observed that its entropy (required bits per weight) distribution spans from 4.5 to 5.5. Since the original weights are represented with 32 bits per weight, the entropy distribution indicates rooms for compression. When we plot the error sensitivity (gradient) of the weights together [Figure 3(c)], it can be seen that the areas with higher error sensitivity tend to have higher entropy. Consequently, the image encoding will inherently assign more bits to areas that are relatively more critical for accuracy.

Although image encoding techniques can help preserve the critical information, loss of information is inevitable because of the quantization process (DCT in JPEG) in the techniques. The weights encoded and decoded by the JPEG algorithm with a single QF value of 20 [Figure 3(d)] shows that high-gradient areas are also distorted. To minimize the distortion of the important areas, we can use an adaptive image compression technique commonly used for region-of-interest image coding [20]. We propose to use a higher QF value for accuracy-critical regions, and a lower QF value for less critical regions. By differentiating the quality factor, blocks with higher sensitivity can be quantized less for higher accuracy, while those with lower sensitivity can be compressed more for higher compression [Figure 3(e)].

B. Adaptive Weight Compression Technique

Overall encoding and decoding process is described in Figure 4. First, the weight values are scaled to [0, 256] to make them grayscale pixel values for an JPEG encoder. Then, every 64 elements of each row of the 2-D weight matrix are reshaped into 8x8 sub-blocks. We encode the weights in a row-wise manner so that a row of decoded weights can be available to a multiply-accumulate (MAC) unit for a single neuron computation. For each corresponding block, we also calculate the average error sensitivity (average absolute value of gradient), which is used to determine the QF of the block. If the average error sensitivity higher than the predefined threshold, high QF value is chosen for quantization, while low QF value is used for blocks with average error sensitivity lower than the threshold. Therefore, the threshold and high/low QF values can be used as knobs to achieve the desired compression ratio and accuracy level. Using the determined QF value, the reshaped input weights are compressed into bit-stream, and stored in the memory of a mobile device for neural network inference.

In addition to the compressed bit-stream of the weights, we need to store a QF map that represents the QF selection of each block [one bit per block (i.e. 64 weights)]. We also need to store the high/low QF values (14 bits/layer) since the decoder requires QF value information of each block for correct dequantization. In this work, we use only two QF values to minimize the overhead of additional bits for representing the QF information.

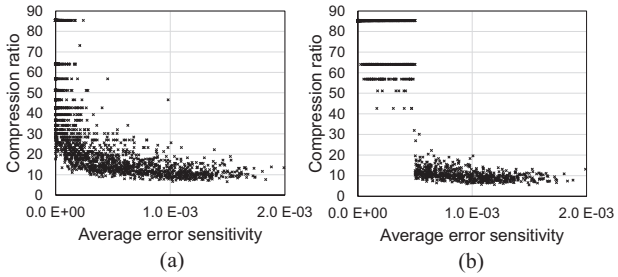

Figure 5. Compression ratio vs. average error sensitivity after JPEG compression with (a) single QF and (b) multi QF.

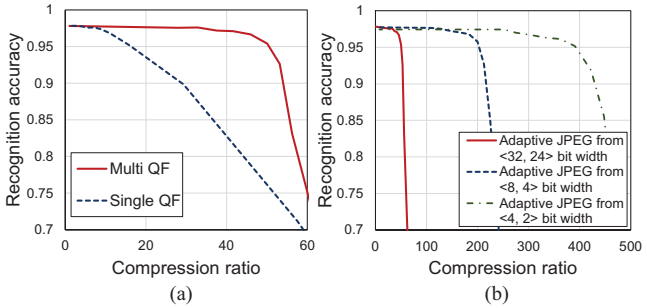

Figure 6. Compression performance of (a) JPEG with single/multi QF, (b) adaptive JPEG compression combined with bit truncation.

At the hardware device for neural network inference, one block of compressed weights is loaded into the on-chip buffer, together with the QF information. The block is decoded by a JPEG decoder and reshaped into the original form of 64-element row vector, which is fed into a MAC unit. Here, we assume the operations (off-chip memory access, JPEG decoding, and MAC operations) are pipelined in a unit of an 8x8 block, to improve the throughput. The major overhead added to the inference device for adaptive image compression is the JPEG decoder. Note the time/energy complexity of JPEG decoder is much lower than encoder as in common image encoding algorithms [21].

V. Performance Analysis

A. Dataset

To validate the proposed method, we first use a typical MLP-based neural network with three benchmark datasets: MNIST [5], CNAE-9 [22], and SVHN [23]. The network has three layers for MNIST (784-144-10), four layers for CNAE-9 (856-144-64-9) and SVHN (1024-144-64-10). Weights in these networks are represented with 32-bit fixed-point precision. The baseline accuracy and weight size of each dataset are described in Table I.

B. Performance of the Adaptive Compression Technique

To visualize the effect of the adaptive image compression approach, we plot average error sensitivity (gradient) and compression ratio of the blocks in the weight matrix of MLP for MNIST. When we use JPEG compression with a single QF value (QF=20), blocks with lower error sensitivity (i.e. lower gradient) tend to be compressed more [Figure 5(a)]. By applying JPEG encoding with two QF values (high QF=25, low QF=5, and threshold = 0.0005), less error-sensitive blocks are compressed even more, while sensitive blocks are compressed less to achieve higher overall compression ratio at the same accuracy (0.9) [Figure 5(b)]. On the other hand, at the same compression ratio, the adaptive approach achieves higher accuracy by preserving weights with higher error sensitivity.

By reducing the encoded bits for error-insensitive blocks, the adaptive JPEG encoding method achieves 42X compression (4.3X more compression than the single-QF method) at 1% loss of normalized accuracy [Figure 6(a)]. The proposed approach can be combined with a bit truncation technique to further compress the weights by reducing the number of bits per weight before applying JPEG encoding [Figure 6(b)]. Reducing the bit-width of the weights also helps decrease the system energy, as a reduced-bit JPEG decoder can be used.

C. Comparison of Compression Performance

For performance comparison on the MLP-based network, we consider four other compression methods: lossless coding based on run-length encoding and Huffman coding (RLE+Huffman) [4][13], dropout of high frequency components of the weights (DropFreq) [14], pruning [4][11], and truncation [10]. For all compression methods, we do not perform fine-tuning after compression. Figure 7(a) shows that the proposed approach achieves higher compression under given accuracy for MNIST dataset. Table I shows compression

Table I. Compression ratio of different compression methods and datasets at the normalized accuracy loss=1%

Dataset	Baseline accuracy	Layer	Size (KB)	Compression ratio						
				RLE+ Huffman[4][13]	DropFreq [14]	Pruning [4][11]	Truncation [10]	Proposed		
								Single QF	Multi QF	Multi QF + truncation
MNIST	0.978	FC1	441.0	1.1	1.9	5.9	8	9.1	42.7	275.0
		FC2	5.6	1.1	1.8	2.9	8	4.6	19.8	115.2
		Total	446.6	1.1	1.8	5.9	8	9.0	42.4	273.0
CNAE-9	0.972	FC1	481.5	1.1	1.8	4.5	8	5.6	19.9	82.6
		FC2	36.0	1.1	1.0	3.2	8	5.1	11.6	69.3
		FC3	2.3	1.0	2.2	2.1	8	3.0	4.6	18.1
		Total	519.8	1.1	1.7	4.4	8	5.6	19.3	81.4
SVHN	0.769	FC1	576.0	1.2	1.5	3.1	4	8.1	26.8	101.3
		FC2	36.0	1.1	1.1	1.6	4	2.4	5.2	20.0
		FC3	2.5	1.1	1.3	1.9	4	2.1	3.6	15.9
		Total	614.5	1.2	1.5	3.0	4	7.8	25.4	96.2

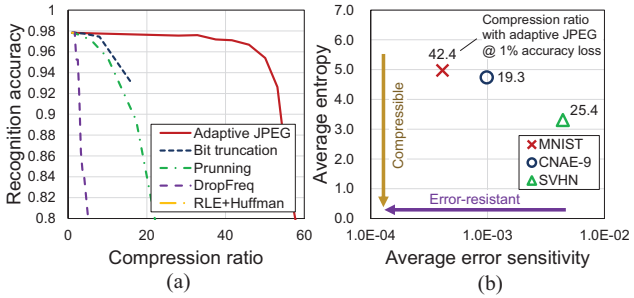

Figure 7. Comparison of compression performance with (a) other approaches and (b) and different datasets.

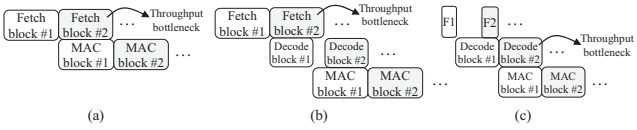

Figure 8. (a) Pipelining diagram without JPEG decoding. (b) Pipelining diagram with JPEG decoding when memory access is bottleneck and (c) decoding is bottleneck.

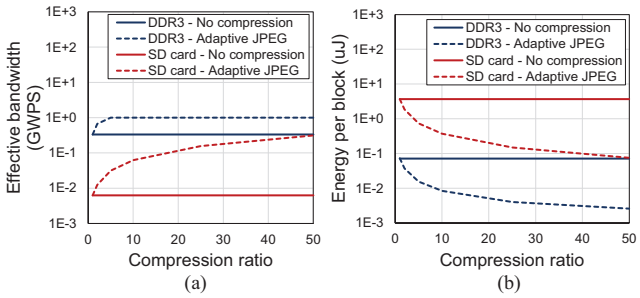

Figure 9. (a) Effective bandwidth and (b) energy per block with compression ratio.

ratio achieved by different approaches at 1% loss of normalized accuracy. For the MNIST dataset, the proposed adaptive QF control approach achieves 42.4X compression, which is significantly higher than other approaches. When combined with the bit truncation, the weight size reduces by 273X, from 446KB to 1.6 KB. When assuming no accuracy loss, it provides 22X reduction, which is comparable performance to the existing approach with fine tuning [4] (40X reduction on a similar 4-layer MLP).

As the performance and energy of the system with the proposed technique depend on the compression ratio of the given trained weights, it would be beneficial to have an estimation of the compression ratio before we actually apply compression. According to the entropy theory, images with lower entropy tend to be compressed more. On the other hand, absolute gradient indicates the sensitivity of accuracy to the changes in weight values. Therefore, a pair of entropy and gradient of the weights matrix can be used to estimate the compression ratio with the accuracy constraint. The gradient-entropy pair and compression ratio of the three datasets on MLP are shown in Figure 7(b). It can be observed that the dataset with lower absolute gradient and entropy tend achieve higher compression ratio by the proposed approach. For example, compression ratio of SVHN is lower than MNIST even with lower average entropy, since its weights are far more error-sensitive.

TABLE II. THROUGHPUT AND ENERGY OF DIFFERENT SYSTEMS

Off-chip memory		DDR3			SD card		
Compression		No	Pruning	Adaptive JPEG	No	Pruning	Adaptive JPEG
Compression ratio		1	5.9	**42.4**	1	5.9	**42.4**
Throughput (GWPS)		0.33	1.95	1	0.006	0.037	0.27
Throughput improvement		-	**5.9**	3	-	5.9	**42.4**
Energy per block (nJ)	Off-chip mem	71.7	12.1	1.7	3686	624.9	86.9
	Decoder	-	-	1.15	-	-	1.15
	MAC	1.03	1.03	1.03	1.03	1.03	1.03
	Total	72.7	13.2	3.9	3687	626	89
Energy reduction		-	5.5	**18.7**	-	5.9	**41.3**

TABLE III. COMPARISON OF THROUGHPUT AND ENERGY IMPROVEMENT

Off-chip memory	Dataset	Compression	Comp. ratio	Throughput improvement	Energy reduction
DDR3	MNIST	Pruning	5.9	**5.9**	5.5
		Adaptive JPEG	**42.4**	3	**18.7**
	CNAE-9	Pruning	4.4	**4.4**	4.2
		Adaptive JPEG	**19.3**	3	**12.3**
	SVHN	Pruning	3	**3**	2.9
		Adaptive JPEG	**25.4**	**3**	**14.5**
SD card	MNIST	Pruning	5.9	5.9	5.9
		Adaptive JPEG	**42.4**	**42.4**	**41.3**
	CNAE-9	Pruning	4.4	4.4	4.4
		Adaptive JPEG	**19.3**	**19.3**	**19.1**
	SVHN	Pruning	3	3	3
		Adaptive JPEG	**25.4**	**25.4**	**25**

VI. SYSTEM PERFORMANCE AND ENERGY ANALYSIS

A. Compression Ratio vs. Memory Peformance and Energy

As compression reduces the memory size required to store the trained weights, memory access delay and energy during inference will decrease accordingly. However, the proposed compression approach requires JPEG decoding, which incurs additional computation time and energy. Therefore, to achieve the advantage in system throughput and energy, compression ratio should be high enough so that the reduction in memory access time and energy can compensate for decoding overhead. As the memory access time and energy vary depending on the off-chip memory type, we perform the throughput and energy analysis with two different memory types: DDR3 [6] and SD card [24]. The reason for considering SD card, a non-volatile memory (NVM), is to realize the fact that if the weights are not updated regularly on-line, storing the weights in NVM can significantly reduce the storage energy. However, SD cards have higher access time/energy than DDR3. The access latency and energy of DDR3 are 0.19 ns/bit, and 70 pJ/bit [25], and 10 ns/bit and 3600 pJ/bit for SD card [24]. Note the memory demand for the datasets presented here can in principle be stored in an on-chip memory as they are based on relatively small images. However, as discussed in Figure 1, even for the same applications more practical image sizes will make on-chip storage impossible. Therefore, for our analysis we have considered an off-chip storage for all applications, with and without compression.

We first analyze the throughput (performance) between the off-chip memory and MAC units, indicating how many synaptic weights can be supplied to MAC units per unit time (GWPS: Giga weights per second). Here we assume block-

level pipelining between an off-chip memory, decoder, and MAC units. In the baseline system without compression, the effective bandwidth is limited by the off-chip memory access time of a block [Figure 8(a)]. As access time for a block decreases due to the adaptive JPEG compression, the effective memory throughput increases even considering the decoding time [Figure 8(b) and Figure 9(a)]. Note that pipelining fetch and JPEG decoding stages helps improve the performance. As compression ratio increases, reduced block access time enhances the throughput until the decoding time becomes the throughput bottleneck [Figure 8(c)]. Therefore, the system throughput saturates at higher compression as illustrated in Figure 9(a). The effect is more pronounced for DDR3 with higher memory performance, compared to the SD card case. To analyze the energy advantage, we illustrate the sum of memory access and decoding energy for given compression ratio [Figure 9(b)]. For all memory types, compression leads to significant reduction in energy because of reduced memory access energy.

B. System Performance and Energy Analysis

For system-level performance and energy analysis, we design a hardware for neural network inference, and synthesize it in 28nm CMOS technology [Figure 4(b)-(c)]. The on-chip system includes 144 MAC units, a JPEG decoder, and two block buffers for block-level pipelining. Table II shows the throughput and energy of the proposed approach and pruning for the MNIST dataset, and Table III compares the throughput and energy improvement for different datasets and off-chip memory types. For relatively lower latency memory (DDR3), high compression ratio of the proposed method does not translate into effective throughout improvement due to decoding time overhead. However, when the off-chip memory access latency is significantly larger than decoding (SD card), the proposed approach achieves much higher throughput improvement than pruning, with its high compression ratio. Also, its higher compression results in significant reduction in system energy even with decoding energy overhead. The energy-efficiency improvement becomes higher when a system uses SD card, where the system energy is dominated by access energy.

VII. Conclusion

We have presented a technique for compressing weights of neural networks using JPEG image encoding. The presented approach exploits the smoothness of weight matrix in a multi-layer perceptron network. By adaptively determining the quantization level of JPEG based on the error-sensitivity of the weights, we show that the proposed approach achieves high compression ratio while preserving error-sensitive information. The high compression leads to the throughput improvement of the inference hardware system at significantly lower energy even with accounting for the JPEG decoding overhead. The proposed compression technique also helps simplify the hardware design process, as it does not involve the training process. Therefore, proposed method can be a simple solution for a resource-constrained hardware with a trained network for inference. The future work needs to consider the design and prototype of the inference engine, as well as more complex feed-forward and recurrent networks.

Acknowledgment

This work was supported by Office of Naval Research Young Investigator Program (2012) and National Science Foundation Career Award (CNS# 1054429).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[2] P. Panda, A. Sengupta, and K. Roy, "Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition Priyadarshini," *DATE*, 2016.

[3] Pavlo Molchanov et al., "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks," *CVPR*, 2016.

[4] S. Han, H. Mao, and W. J. Dally, "Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *ICLR*, 2016.

[5] "MNIST database." [Online]. http://yann.lecun.com/exdb/mnist/.

[6] "DDR3 SDRAM, JESD79-3F." [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd-79-3d.

[7] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube : A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *ISCA*, 2016.

[8] J. Kung, D. Kim, and S. Mukhopadhyay, "Dynamic Approximation with Feedback Control for Energy-Efficient Recurrent Neural Network Hardware," *ISLPED*, 2016.

[9] M. Courbariaux and Y. Bengio, "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv*, p. 9, 2016.

[10] M. Courbariaux, Y. Bengio, and J.-P. David, "Low Precision Storage for Deep Learning," *ICLR*, 2015.

[11] J. A. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. 1991.

[12] B. R. Tara N . Sainath , Brian Kingsbury , Vikas Sindhwani , Ebru Arisoy, "Low-Rank Matrix Factorization for Deep Neural Network Training With High-Dimensional Output Targets," *ICASSP*, 2013.

[13] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *ISSCC*, pp. 262–263, 2016.

[14] J. Koutník, F. Gomez, and J. Schmidhuber, "Evolving Neural Networks in Compressed Weight Space," *Proc. 12th Annu. Conf. Genet. Evol. Comput. - GECCO '10*, p. 619, 2010.

[15] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Convolutional Neural Networks in the Frequency Domain," *KDD*, 2016.

[16] J. Chung and T. Shin, "Simplifying Deep Neural Networks for Neuromorphic Architectures," in *DAC*, 2016.

[17] J. Kung, D. Kim, and S. Mukhopadhyay, "A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses," *ISLPED*, 2015.

[18] "Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)."

[19] N. Tavakoli, "Entropy and Image Compression," *J. Vis. Commun. Image Represent.*, 1993.

[20] Y. Hu, F. Meng, and Y. Wang, "Improved JPEG Compression Algorithm Based on Saliency Maps," *CISP*, 2012.

[21] L. Sun, Z. Lin, F. Xiao, and J. Guo, "A New Scheme Based on Pixel-Intense Motion Block Algorithm for Residual Distributed Video Coding," *Int. J. Distrib. Sens. Networks*, 2013.

[22] "CNAE-9 Dataset." https://archive.ics.uci.edu/ml/datasets/CNAE-9.

[23] Y. Netzer and T. Wang, "Reading digits in natural images with unsupervised feature learning," *NIPS*, 2011.

[24] "SD Specifications Version 4.10." SD Association.

[25] K. T. Malladi, "Towards energy proportional datacenter memory with mobile dram," *ACM SIGARCH Comput. Archit. News*, 2012.