HardCompress: A Novel Hardware-based Low-power Compression Scheme for DNN Accelerators

Ayush Arunachalam*, Shamik Kundu*, Arnab Raha†, Suvadeep Banerjee†, Surajyaprasad Natarajan† and Kanad Basu*

*University of Texas at Dallas, Richardson, TX 75080
†Intel Corporation, Santa Clara, CA 95054

Abstract—The ever-increasing computing requirements of Deep Neural Networks (DNNs) have accentuated the deployment of such networks on hardware accelerators. Inference execution of large DNNs often manifests as an energy bottleneck in such accelerators, especially when used in resource-constrained Internet-of-Things (IoT) edge devices. This can be primarily attributed to the massive energy incurred in accessing millions of trained parameters stored in the on-chip memory, as demonstrated in existing research. To address this challenge, we propose HardCompress, which, to the best of our knowledge, is the first compression solution pertaining to commercial DNN accelerators. The three-step approach involves hardware-based post-quantization trimming of weights, followed by dictionary-based compression of the weights and subsequent decompression by a low-power hardware engine during inference in the accelerator. The efficiency of our proposed approach is evaluated on both lightweight networks trained on MNIST dataset and large DNNs trained on ImageNet dataset. Our results demonstrate that HardCompress, without any loss in accuracy on large DNNs, furnishes a maximum compression of 99.27%, equivalent to $137\times$ reduction in memory footprint in the systolic array-based DNN accelerator.

Index Terms—DNN accelerator, Hardware-based trimming, Dictionary-based Compression, Low-power Decompression

I. INTRODUCTION

Machine Learning (ML) and Artificial Intelligence (AI) are being extensively adopted to solve a host of real-world problems ranging from object detection and face recognition to natural language processing and data mining. Traditional CPU- and GPU-based deep learning implementations incur high overhead in terms of latency, computation and power. To address these challenges, customized DNN accelerators are being designed by both semiconductor and software organizations. Google’s Tensor Processing Unit (TPU) is one such DNN accelerator, which furnishes $15 \times - 30 \times$ better performance than conventional CPUs and GPUs [1].

Reduction in energy overhead is imperative to facilitate application of such DNN accelerators in resource-constrained environments. Towards this end, techniques such as approximate computing [2]–[4], and software-hardware co-design approaches that consider sparsity of weights have been proposed [5]. However, none of these methods focus on reducing the enormous memory footprint arising from millions of trained DNN parameters, accessing which incurs massive energy overhead in such accelerators [5]–[7].

To address this challenge, we propose HardCompress, a three-step approach involving hardware-based trimming of weights, dictionary-based compression, and low-power decompression in hardware, as outlined in Figure 1. While quantization and dictionary-based compression are performed offline, hardware-based weight trimming, memory fetching of weights, and decompression are done online. The key contributions of this paper are:

- This paper proposes HardCompress, which, to the best of our knowledge, is the first hardware-based low-power deep compression solution for DNN accelerators deployed in resource-constrained IoT edge devices.
- We propose hardware-based post-quantization trimming of weights, facilitating the removal of 1s in addition to 0s.
- We propose lossy and lossless dictionary-based compression schemes to optimize the trimmed weight memory, and a novel Euclidean distance-based dictionary selection algorithm for the lossy compression technique.
- We develop a low-power hardware decompression engine and determine the area and power overhead, by synthesizing it using Synopsys Design Compiler.
- The proposed HardCompress scheme has been evaluated on both lightweight and popular deep neural networks.

The rest of the paper is organized as follows. Section II explores related work in deep compression. The fundamentals of DNN accelerators and dictionary-based compression have been outlined in Section III. Section IV motivates the proposed approach, HardCompress, which is explained in detail in Section V. Section VI analyzes the performance of HardCompress on different DNNs. Finally, the paper is concluded in Section VII.

II. RELATED WORK

In this paper, we have developed HardCompress, a hardware-based low-power deep compression solution for DNN accelerators deployed in resource-constrained IoT edge devices. In order to improve energy efficiency in DNN accelerators, researchers have proposed approximate computing-based solutions. They aim to abate the energy requirements through modification in the design datapath with acceptable degradation in computation accuracy [3], [8], [9]. However,
approximate computing does not address the energy overhead due to memory access, which constitutes a considerable percentage of the total energy requirement during inference [5]–[7]. In this direction, TraNNsformer, a hardware-based pruning approach for memristive training hardware, was proposed by [10]. Unlike TraNNsformer, HardCompress focuses on CMOS-based inference networks. Moreover, while TraNNsformer performs pruning without quantization, HardCompress applies a post-quantization weight trimming technique that facilitates the removal of both 0 and 1 weight values. Finally, while TraNNsformer restricts itself to pruning, HardCompress uses a compression scheme to reduce the size of the trimmed weights.

Recently, an inference engine, EIE, was proposed for software-based deep compression techniques [7]. HardCompress is fundamentally different from EIE in the following ways: (1) HardCompress, to the best of our knowledge, is the first deep compression solution for commercial DNN accelerators. It can be adopted in any generic off-the-shelf systolic array-based DNN accelerator, unlike EIE, which is a dedicated inference engine for software-based deep compression techniques [11], [7] uses features like weight-sharing to reduce energy footprint, which may not be available in any generic off-the-shelf DNN accelerator. It should be mentioned that performance of HardCompress can be augmented when implemented in an accelerator with weight-sharing feature, leading to further compression in memory. (2) Compared to [11], our post-quantization weight trimming technique is able to remove weights of value 1, in addition to value 0. Since weights of value 1 occupy a considerable portion of the total weight array (as seen in Table 1), this significantly improves the compression performance. (3) Since we primarily focus on formulating a low-power solution for IoT edge devices, we incorporate dictionary-based compression algorithm in our solution. In comparison to Huffman coding used by [11], dictionary-based compression provides lower decompression overhead in terms of area and power, as shown by [12]. We have also validated the claims in Section IV-B.

III. BACKGROUND

A. DNN Accelerators

DNN inference consists of computationally complex matrix multiplication operations. These operations are accelerated using a mesh-like topology called systolic array, which is used extensively as an integral part of the architecture of DNN accelerators [1]. Multiplier-Accumulator (MAC) units, the fundamental elements of a systolic array, are responsible for the multiplication and accumulation operations that DNN inference entails. The representative architecture of an $N \times N$ dimension systolic array having $N^2$ MAC units is shown in Fig. 2. The pre-trained weights are fetched from the memory of the accelerator and loaded into the systolic array such that weight $w_{ij}$ is mapped to $MAC_{i,j}$ ($i$ and $j$ represent the row and column indices, respectively). Increase in the size of DNNs results in an increase in the number of weights being fetched from memory, which leads to a larger memory footprint, and thus, energy inefficiency [5], [6]. The memory access energy of the weights is directly proportional to the number of bits used to represent them. Hence, the technique of quantization (which reduces the number of bits

![Fig. 2: Representative architecture of a DNN accelerator.](image)

representing each value) is the first step towards optimizing the memory footprint and energy consumption of the DNN execution platform [5], [7], [11].

B. Dictionary-based Compression

HardCompress uses dictionary-based compression, which entails using a dictionary to represent a set of values. It performs memory compression by defining a dictionary of length $L$ to represent all $N$ values belonging to a set of weights, where $L < N$. The goal of the dictionary is to compress maximum number of weights. The dictionary selection algorithm can differ depending on the application [12]–[14].

Illustrative Example: There are two types of dictionary-based compression — lossless and lossy. Lossless compression does not incur any reduction in accuracy; however, the compression performance is not guaranteed. On the other hand, lossy compression ensures a certain fixed compression, but is accompanied by a reduction in accuracy. Both compression schemes are explained in Figure 3, which considers a set of ten 8-bit values, with the total size of the set being 80 bits. It depicts the original values and the corresponding lossy and lossless compressed values. Here, we have a dictionary of length 2, which can be represented using only 1 bit, with the two most frequently occurring values being selected as the dictionary entries. In lossy compression, when there is a match, the original value is replaced by the corresponding dictionary index. In the event of a mismatch, the value is approximated by the closest dictionary entry, in terms of Euclidean distance. For example, the third entry, 00001111, is closest in value to the first dictionary entry, i.e., 00000101, and hence is represented by the dictionary index 0. Entry 9 corresponds to the second dictionary entry and hence, we represent it as 1. After compression, the size of the set reduces to 10, and hence, a memory reduction of 87.5% is obtained. Since the original $b$-bit weights are represented by $k$ bits, where $k = \log_2 L$ and $L$ is the length of the dictionary, lossy compression furnishes a fixed compression of $1 - (k/b)$. However, as the values of the mismatched weights are being replaced, these approximations tend to cause degradation in accuracy. For lossless compression, in the event of a match with the dictionary, the MSB is set to 0, followed by the corresponding dictionary index in the LSB. In the event of a mismatch, the MSB is set to 1, followed by the original 8-bit value. For example, the third value, 00001111, is not present in the dictionary and is now appended with a 1 in the MSB location, forcing the bit length to 9. However, the seventh and ninth entries are present in the dictionary and are now compressed to 2 bits. The total size of the set reduces
Fig. 3: Illustrative example of dictionary-based compression to 55 bits, which yields a compression of only 31.25%. Although lossless compression induces no loss in accuracy, the compression performance is not guaranteed.

IV. MOTIVATION

Large DNN models have high energy overhead because of the enormous memory footprint that is associated with fetching the trained parameters. Existing research has shown that the memory access energy associated with on-chip SRAM is lower than off-chip DRAM. While SRAM requires 5 pJ of energy for a single fetch of a 32-bit weight, DRAM furnishes 128× higher energy consumption of 640 pJ for the identical operation [7]. However, since SRAM is considerably smaller in size compared to DRAM, the trained parameters of large DNNs are stored in the off-chip DRAM. Fetching these parameters from DRAM is highly energy and power intensive. For instance, consider a DNN with 700M parameters running at 20 Hz. With power being defined as time rate of transferring energy, the power required to fetch this DNN from DRAM would be 20×700M×640 pJ = 8.96W, which is too high for resource-constrained devices. HardCompress addresses this problem by performing hardware-based trimming of weights, followed by dictionary-based compression, which facilitates the storing of DNN models in SRAM. This section describes the motivation behind these two approaches.

A. Post-Quantization Hardware-based Weight Trimming

The distribution of weights in large CNN models, such as VGG-16, MobileNet, and ResNet50 is biased, with lower-value weights being predominant [11]. In our framework, we assume the weights are quantized before being loaded into the memory, similar to [15]–[17]. As shown in Table I, the large DNNs that are trained on ImageNet have a considerable number of 0 and 1 weight values post-quantization, with MobileNet having the highest of 98.73%. This motivates us to perform post-quantization trimming of both 0 and 1 weight values in hardware, to reduce the memory requirement.

<table>
<thead>
<tr>
<th>Model</th>
<th>Fraction of 0s (%)</th>
<th>Fraction of 1s (%)</th>
<th>Fraction of 0s and 1s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>98.93</td>
<td>1.80</td>
<td>98.93</td>
</tr>
<tr>
<td>VGG-16</td>
<td>64.86</td>
<td>23.52</td>
<td>88.38</td>
</tr>
<tr>
<td>VGG-19</td>
<td>62.85</td>
<td>24.13</td>
<td>86.98</td>
</tr>
<tr>
<td>ResNet50</td>
<td>21.12</td>
<td>19.25</td>
<td>40.36</td>
</tr>
</tbody>
</table>

B. Dictionary-based Compression

In our approach, the weights are compressed once, i.e., offline, and stored in the memory of a systolic array-based DNN accelerator. However, decomposition is done dynamically, i.e., online, where the compressed weights are mapped back to their original value in the MAC units during DNN inference. Therefore, power and area overhead of the decompression engine must be minimized for HardCompress to qualify as a low-power solution.

Huffman Coding, which is used in software-based deep compression techniques [11], has a larger area and power overhead compared to dictionary-based compression technique, as shown in existing research [12]. In order to validate this claim for our specific application environment, we designed the Register-Transfer Levels (RTL) for both the dictionary-based decompressor and Huffman decoder engines. Next, we synthesized them using Synopsys Design Compiler logic synthesis tool and 1st_10k cell library. Table II describes the area and power overhead of both hardware with respect to the 256×256 dimension systolic array of the accelerator for ResNet50. The corresponding results demonstrate that the area and power overhead of the dictionary-based decomposition hardware is 15× and 71× lesser than that of Huffman decoder hardware. Since HardCompress is directed towards low-power IoT edge devices, dictionary-based compression is preferred over Huffman Coding.

V. METHODOLOGY

This section describes our proposed compression technique, HardCompress, which consists of three steps, as explained in the following subsections.

A. Post-Quantization Hardware-based Weight Trimming

Figure 4 depicts hardware-based weight trimming, the first step of HardCompress, in a weight-stationary systolic array-based DNN accelerator. HardCompress assumes the quantized weight values are stored in the memory of the accelerator, similar to [15]–[17]. If a weight of value 0 is encountered, the corresponding MAC unit is bypassed, since, passing a value of 0 to a MAC unit is equivalent to disabling it. In Figure 4, \( MAC_{1,N} \), which is mapped to weight \( w_{1N} = 0 \), is bypassed. In case of a weight of value 1, we bypass the multiplier of the corresponding MAC unit, making it function only as an adder. Since the weight is 1, the result of the multiplier is identical to the input activation. Therefore, we add the input activation directly to the output of the previous MAC unit in the same column. In Figure 4, \( MAC_{2,N} \) is mapped to weight \( w_{2N} = 1 \). We bypass the multiplier of \( MAC_{2,N} \) and add its input \( a_2 \) to the output of
MAC$_{1,N}$, resulting in $a_2$. Hence, weights of values 0 and 1 are not required to be stored in the weight memory. Since these weights comprise a considerable portion of the large DNN networks, as outlined in Table I, this step provides substantial reduction in memory traffic.

B. Dictionary-based Compression

The second step of HardCompress involves dictionary-based compression, which is implemented offline. For lossless compression, the most frequently occurring weight values are selected as the dictionary entries, similar to [14]. However, for lossy compression, the dictionary selection algorithm needs to incorporate the correlation (based on Euclidean distance) between mismatched weights and dictionary entries as well, as shown in Section III-B. Hence, we have developed an algorithm, as discussed in this section, which considers both frequency of weights and approximation using Euclidean distance towards dictionary selection.

The proposed dictionary selection approach is outlined in Algorithm 1. The first input, $absWeights$, is the set of absolute values of the weights, which are stored in the memory of the DNN accelerator. The other two inputs, quantization values of the weights, which are stored in the memory $absWeights$

### Algorithm 1 Lossy Compression-Aware Dictionary Selection

**Input:** $absWeights$, $b, L$

**Output:** $dictVals$

1. Initialize $dictVals \leftarrow$ empty Array
2. for $L$ iterations do
   3. for $i$ in range$(0, 2^b - 1)$ do
      4. Initialize $choiceArray[i] \leftarrow 0$
      5. for $j$ in $absWeights$ do
         6. $choiceArray[i] += abs(i - j)$
      7. end for
   8. end for
   9. $minValue \leftarrow \min(choiceArray)$
10. for $k$ in $choiceArray$ do
      11. if $minValue == choiceArray[k]$ then
         12. $dictVals \leftarrow k$
         13. break
      14. end if
15. end for
16. if $dictVals$ already has the index $k$ then
   17. for $i$ in $dictVals$ do
      18. $choiceArray[i] += \max(choiceArray)$
   19. end for
20. goto 9
21. end if
22. end for

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

HardCompress is evaluated using two different sets of experiments, (1) on lightweight Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) (trained on MNIST dataset) and (2) on large CNNs —VGG-16, VGG-19, MobileNet, and ResNet50 (trained on ImageNet dataset [18]). The lightweight MLP has 4 fully connected layers of $[256 - 256 - 256 - 10]$ configuration. The lightweight CNN comprises 3 convolutional layers, with a max pooling layer following each of the first two. The size of the feature maps are $[26 \times 26 \times 32, 13 \times 13 \times 32, 11 \times 11 \times 64, 5 \times 5 \times 64, 3 \times 3 \times 64]$. A flattening layer and two fully connected layers of dimension $[576, 64, 10]$ form the terminal layers of this CNN model. The network parameters and activations of the DNNs are quantized to 8 bits$^{1}$, similar to [15], and subsequently mapped to the DNN accelerator. In both experiments, post-weight trimming, the number of dictionary entries are varied in incremental powers of 2. If the length of the original quantized weights is $b$ bits, then the compressed weights must have a length of $b - 1$ bits or shorter to furnish compression. Therefore, the corresponding maximum dictionary length is $L = 2^b - 1$ for lossy compression. However, lossless compression incorporates an additional bit to represent the compressed or uncompressed nature of the weights. Hence, the compressed weights (without the MSB) must have a length of $b - 2$ bits or shorter to furnish compression, which corresponds to a maximum dictionary length of $L = 2^{b-2}$. Since the network parameters are quantized to 8 bits, i.e., $b = 8$.

$^1$The weights can be quantized to other values, like 16-bits, 32-bits as well. We have used 8-bits for our experimental purpose.
\(b = 8\), the maximum dictionary length that still guarantees compression is \(L = 128\) and \(L = 64\) for the lossy and lossless schemes, respectively. Therefore, to maintain parity, the dictionary length \(L\) for both lossy and lossless compression is varied from 1 to 64, in incremental powers of 2.

B. Results

1) Lightweight MLP and CNN: The results for the lightweight networks are shown in Table III and Figure 5. Table III describes the compression results for both MLP and CNN, as outlined in column 1. The second column depicts the results of hardware-based weight trimming. Columns 3 and 4 demonstrate the compression obtained using lossy and lossless schemes, respectively. The compression values for the lossy approach are reported for an optimal dictionary length that furnishes substantial compression performance in lieu of graceful degradation in classification accuracy. While a dictionary of unit length will exhibit the best compression, such a dictionary is not appropriate, since it would induce massive penalty on the classification accuracy of the network. On the other hand, as the lossless approach does not degrade the classification accuracy, the compression values reported in column 4 correspond to the dictionary length that furnishes the best compression performance for the lossless scheme. The last column of the table outlines the improvement in compression performance of the lossy scheme over lossless.

Figure 5a illustrates the variation of compression with dictionary length \(L\) for the lossless scheme, while Figure 5b exhibits the improvement in classification accuracy with increase in dictionary length for lossy compression.

As seen in Table III, the initial step of hardware-based trimming of weights furnishes compression of 5.97% and 4.98% for MLP and CNN, respectively. The low compression values can be attributed to the minor presence of weights 0 and 1, which is lower than that in the larger models. Post weight trimming, the lossy approach furnishes a maximum compression of only 13.49% for MLP at \(L = 32\) (10.77% for CNN at \(L = 64\)), as represented in Figure 5a. On the other hand, the lossy technique yields an accuracy of 95.09% (against a baseline accuracy of 97.66%) in the MLP, for a compression of 25% at \(L = 64\), as illustrated in Figure 5b.

The lightweight CNN furnishes the best accuracy score of 96.61% at \(L = 64\) against a baseline accuracy of 99.01% using lossy compression. From the results, it is evident that there exists a trade-off between compression and accuracy for the lossy scheme. For example, consider the MLP model. Although \(L = 64\) furnishes an accuracy of 95.09% compared to 87.5% furnished by \(L = 8\), the compression is a meagre 25% compared to 62.5% furnished by the latter. Furthermore, the compression scheme also plays a vital role in the overall performance of HardCompress, as evidenced by columns 3, 4 and 5 of Table III. While the lossy scheme furnishes compression as high as 52.49% for \(L = 16\) and an acceptable accuracy of 94.2% in lightweight CNN, equivalent to 2.1× reduction in memory, the lossless scheme yields a maximum compression of only 15.22% for \(L = 64\), equivalent to 1.18× reduction in memory requirement. Therefore, lossy compression performs 1.78× better than lossless compression in the lightweight CNN. Similarly, it outperforms lossless compression by 1.38× in the lightweight MLP. Overall, lossy compression yields 1.38× to 1.78× higher reduction in memory footprint than lossless compression, as seen in the column 5 of Table III, while inducing around 5% degradation in accuracy. This reduction in memory footprint results in lowered energy consumption and power requirement during inference in the DNN accelerator. Hence, if slight degradation in accuracy is acceptable, we suggest incorporating lossy compression in HardCompress for lightweight networks.

TABLE III: Performance analysis on lightweight models. HT: Hardware-based weight trimming; DC: Dictionary-based Compression. Impr: Improvement of HT+Lossy DC over HT+Lossless DC.

<table>
<thead>
<tr>
<th>Model</th>
<th>HT (%)</th>
<th>HT+Lossy DC (%)</th>
<th>HT+Lossless DC (%)</th>
<th>Impr</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>5.97</td>
<td>41.23</td>
<td>18.65</td>
<td>1.38×</td>
</tr>
<tr>
<td>CNN</td>
<td>4.98</td>
<td>52.49</td>
<td>15.22</td>
<td>1.78×</td>
</tr>
</tbody>
</table>

2) Large CNNs trained on ImageNet: In the second experiment, hardware-based trimming of weights and lossless compression are performed on large CNNs trained on the ImageNet dataset. Table IV outlines the compression results for various CNN models, which are mentioned in the first column. The second column refers to the dictionary length \(L\) that furnishes maximum compression for the lossless approach in each model. Column 3 depicts the compression results of hardware-based weight trimming, and column 4 demonstrates the net compression obtained using HardCompress (due to hardware-based weight trimming and dictionary-based compression). The variation of compression with dictionary length \(L\) for the lossless approach in large CNNs is represented in Figure 6.

TABLE IV: Performance analysis on large CNN architectures trained on ImageNet.

<table>
<thead>
<tr>
<th>Model</th>
<th>Optimum L.</th>
<th>Compression - HT (%)</th>
<th>Compression - HT + DC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>8</td>
<td>88.91</td>
<td>99.27</td>
</tr>
<tr>
<td>VGG16</td>
<td>4</td>
<td>86.98</td>
<td>94.91</td>
</tr>
<tr>
<td>VGG19</td>
<td>4</td>
<td>86.98</td>
<td>94.35</td>
</tr>
<tr>
<td>ResNet50</td>
<td>8</td>
<td>80.36</td>
<td>87.52</td>
</tr>
</tbody>
</table>

As seen in column 3 of Table IV, a maximum compression of 98.73%, which is equivalent to 79× reduction in memory requirement, is achieved for MobileNet, owing to the substantial presence of 0s and 1s in its weight matrix. With at least 86.98% compression in three out of the four models, hardware-based trimming of weights furnishes considerable reduction in memory. As is evident from Figure 6, the lossless scheme furnishes highest post-weight trimming compression of 56.41% and 56.16% at \(L = 4\) for VGG-19 and VGG-16, respectively. The net effect of implementing
hardware-based weight trimming and lossless compression is a significant reduction in memory requirement. As seen in column 4 of Table IV, compression as high as 99.27%, which is equivalent to \(137 \times\) reduction in memory footprint, is obtained in MobileNet. Overall, HardCompress furnishes substantial compression of 94% or higher in three of the four large CNN models. From Table IV, we can infer that the large CNN models used in our experiments require a maximum of \(L = 8\) dictionary entries to produce optimum results. Since, unlike the lightweight models, the implementation of lossless scheme provides substantial compression (at least 94% for three of the four large CNN models) with no loss in accuracy, it is preferred over lossy compression.

C. Decompression Engine and its Overhead

In this experiment, we determine the hardware overhead incurred to implement the decompression engine in the systolic array of the DNN accelerator. For each dictionary length \(L\) as mentioned in Section VI-B, we designed the decompression RTL using Verilog and synthesized using Synopsys Design Compiler logic synthesis tool, utilizing two different digital standard cell libraries, lsi_10k and saed_90nm. The RTL of a single MAC unit in the systolic array is also designed and synthesized similarly. The area and power overheads for each \(L\) are derived as per the discussion in Section V-C, and the corresponding values are outlined in Figures 7a and 7b, respectively. The area and power overheads, although extremely insignificant, consistently increase with the increase in \(L\). The dictionary of length \(L = 64\) incurs a maximum area overhead of 0.025% and 0.022% for lsi_10k and saed_90nm, respectively. Correspondingly, the dictionary furnishes a maximum power overhead of 0.004% for lsi_10k and 0.002% for saed_90nm standard cell libraries. Hence, in lieu of minimal overhead, this hardware engine efficiently decompresses the compressed weights, thereby ensuing a low energy inference execution in the DNN accelerator.

VII. CONCLUSION

This paper proposes HardCompress, a compression scheme that primarily focuses on providing a low-power IoT solution for DNN inference. To the best of our knowledge, this is the first compression strategy pertaining to commercial DNN accelerators. The initial step involves hardware-based trimming of quantized weight values 0 and 1, following which dictionary-based compression is implemented. Finally, the compressed weights are decompressed by a low-power decomposition engine in the inference phase. HardCompress is evaluated using lightweight custom-defined MLP and CNN, trained on MNIST dataset, and larger CNNs such as VGG-16, MobileNet and ResNet50, trained on ImageNet dataset. Hardware-based trimming of weights is highly effective, reducing memory footprint by 98.73% in MobileNet. Overall, HardCompress furnishes compression up to 99.27%, which is equivalent to \(137 \times\) reduction in memory requirement. Moreover, we have designed a low-power hardware decomposition engine that furnishes additional area savings of \(15 \times\) and power savings of \(71 \times\) compared to Huffman Coding, which is used in software-based deep compression techniques. The performance of HardCompress can be further augmented using weight sharing feature, when implemented in customized DNN accelerators.

REFERENCES