Extended Bit-Plane Compression for Convolutional Neural Network Accelerators

Extended Bit-Plane Compression for Convolutional Neural Network Accelerators

Lukas Cavigelli, Luca Benini
Integrated Systems Laboratory, ETH Zurich, Switzerland — {cavigelli, benini}@iis.ee.ethz.ch

***Abstract*—After the tremendous success of convolutional neural networks in image classification, object detection, speech recognition, etc., there is now rising demand for deployment of these compute-intensive ML models on tightly power constrained embedded and mobile systems at low cost as well as for pushing the throughput in data centers. This has triggered a wave of research towards specialized hardware accelerators. Their performance is often constrained by I/O bandwidth and the energy consumption is dominated by I/O transfers to off-chip memory. We introduce and evaluate a novel, hardware-friendly compression scheme for the feature maps present within convolutional neural networks. We show that an average compression ratio of 4.4× relative to uncompressed data and a gain of 60% over existing method can be achieved for ResNet-34 with a compression block requiring <300 bit of sequential cells and minimal combinational logic.**

***Index Terms*—Compression, Deep Learning, Convolutional Neural Networks, Hardware Acceleration**

I. Introduction

Computer vision has become a key ingredient for automatized data analysis over a board range of real-world applications: medical diagnostics [1], industrial quality assurance [2], video surveillance [3], advanced driver assistance systems [4], and many others. Many of these applications have only recently become feasible due to the tremendous increases in accuracy—even surpassing human capabilities [5]—that have come with the rise of deep learning, and particularly, convolutional neural networks (CNNs, ConvNets).

While CNN-based methods often require a significant computational effort, many of these applications should run in real-time on embedded and mobile systems. This has driven the development of specialized platforms, dedicated hardware accelerators, and optimized algorithms to reduce the number of compute operations as well as the precision requirements for the arithmetic operations [6]–[15].

When looking at these hardware platforms, the energy associated with loading and storing intermediate results/feature maps (and gradients during training) in external memory is not only significant, but often clearly higher than the energy used in computation and on-chip data buffering. This is even more striking when looking at networks optimized to reduce the computation energy by quantizing the weights to one bit, two bits, or power-of-two values, thereby eliminating the need for high-precision multiplications [16]–[20].

Many compression methods for CNNs have been proposed over the last few years. However, many of them are focusing exclusively on

1) compressing the parameters/weights, which make up only a small share of the energy-intensive off-chip communication [21]–[23],
2) exploiting the sparsity of intermediate results, which is not always present (e.g. in partial results of a convolution layer or otherwise before the activation function is applied) and is not optimal in the sense that the non-uniform value distribution is not capitalized [24]–[26],
3) very complex methods requiring large dictionaries, or otherwise not suitable for a small, energy-efficient hardware implementation—often targeting efficient distribution and storage of trained models to mobile devices or the transmission of intermediate feature maps from/to mobile devices over a costly communication link [23].

In this paper, we propose and evaluate a simple compression scheme for intermediate feature maps that can exploits sparsity as well as the distribution of the remaining values. It is suitable for a very small and energy-efficient implementation in hardware (<300 bit of registers), and could be inserted as a stream (de-)compressor before/after a DMA controller to compress the data by 4.4× for 8 bit AlexNet.

II. Related Work

There are several methods out there describing hardware accelerators which exploit feature map sparsity to reduce computation: Cnvlutin [8], SCNN [9], Cambricon-X [10], NullHop [11], Eyeriss [12], EIE [13]. Their focus is on power gating or skipping some of the operations and memory accesses. While this automatically entails defining a scheme to feed the data into the system, minimizing the bandwidth was not the primary objective of any of them. They all use one of three methods:

1) Zero-RLE (used in SCNN): A simple run-length encoding for the zero values, i.e. a single prefix bit followed by the number of zero-values or the non-zero value.
2) Zero-free neuron array format (ZFNAf) (used in Cnvlutin): Similarly to the widely-used compressed sparse row (CSR) format, non-zero elements are encoded with an offset and their value.
3) Compressed column storage (CCS) format (e.g. used in EIE): Similar to ZFNAf, but the offsets are stored in relative form, thus requiring less bits to store them. Few bits are sufficient, and in case they are all exhausted, a zero-value can be encoded as if it was non-zero.

Most compression methods are focusing on minimizing the model size. Most of them are very complex (area) to implement in hardware and need large dictionaries. One such method, deep compression [23], combines pruning, trained clustering-based quantization, and Huffman coding. Most of

The authors would like to thank *armasuisse Science & Technology* for funding this research. This project was supported in part by the EU's H2020 programme under grant no. 732631 (OPRECOMP).

these steps involved cannot be applied to the intermediate feature map, which change for every inference as opposed to the weights which are static and can be optimized off-line. Furthermore, applying Huffman coding—while being optimal—implies storing a large dictionary (typically several MB). Similar issues arise when using Lempel-Ziv-Welch (LZW) coding [27], [28] as present in e.g. the ZIP compression scheme, where the dictionary is encoded in the compressed data stream. This makes it unsuitable for a lightweight and energy-efficient VLSI implementation [29], [30].

Few more methods exist by changing the CNN's structure in order to compress the weights [21], [22] or the feature maps [25], [26]. However, they require altering the CNN's model and retraining, and they introduce some accuracy loss. Furthermore, they can only be used to compress a few feature maps at specific points within the network and introduce additional compute effort, such as applying a Fourier transform to the feature maps.

The most directly comparable approach, cDMA [24], describes a hardware-friendly compression scheme to reduce the data size of intermediate feature maps. Their target application differs in that their main goal is to allow faster offloading of the feature maps from GPU to CPU memory through the PCIe bandwidth bottleneck during training, thereby enabling larger batch sizes and deeper and wider networks without sacrificing performance. They propose to use *zero-value compression (ZVC)*, which takes a block of 32 activation values, and generates a 32-bit mask where only the bits to the non-zero values are set. The non-zero values are transferred after the mask. This provides the main advantage over Zero-RLE that the resulting data volume is independent of how the values of the feature maps are serialized while also providing small compression ratio advantages. Note that this is a special case of Zero-RLE with a maximum zero burst length of 1.

For this work, we build on a method known in the area of texture compression for GPUs, *bit-plane compression (BPC)* [31], fuse it with sparsity-focused compression methods, and evaluate the resulting compression algorithm on intermediate feature maps to show compression ratios of 4.4× and 2.8× for 8 bit AlexNet and SqueezeNet, respectively.

III. Compression Algorithm

An overview of the proposed algorithm is shown in Fig. 2. We motivate its structure based on the properties we have observed in feature maps (cf. Section IV-B):

1) Sparsity: The value stream is decomposed into a zero/non-zero stream on which we apply run-length encoding to compress the zero burst commonly occurring in the data.

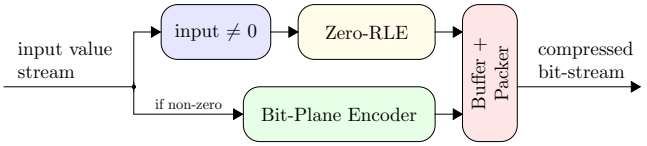

Fig. 2: Top-level view of the proposed compression scheme. The bit-plane encoder is further illustrated in Fig. 1.

2) Smoothness: Spatially neighboring values are typically highly correlated. We thus compress the non-zero values using bit-plane compression. The later compresses a fixed number of words n jointly, and the resulting compressed bit-stream is injected immediately after at least n non-zero values have been compressed.

The resulting algorithm can be viewed as an extension to bit-plane compression to better exploit the sparsity present in most feature maps.

A. Zero/Non-Zero Encoding with RLE

The run-length encoder simply compresses bursts of 0s with a single 0 followed by a fixed number of bits which encode the burst length. Non-zero values, at this point 1-bits, are not run-length encoded, i.e. for each of them a 1 is emitted. If the length of a zero-burst exceeds the corresponding maximum burst length, the maximum is encoded and the remaining bits are encoded independently, i.e. in the next code symbol.

B. Bit-Plane Compression

An overview of the bit-plane compressor (BPC) used to compress the non-zero values is shown in Fig. 1. For BPC a set of n words of m bit, a *data block*, is compressed by first building differences between each two consecutive words and storing the first word as the base. This exploits that neighboring values are often similar to reduce concentrates the distribution of the compressed values around zero.

The data items storing these differences are then viewed as $m+1$ bit-planes of n bit each (delta bit-planes, DBPs). Neighboring DBPs are XOR-ed, now called *DBX*, and the DBP of the most significant bit is kept as the base-DBP. The results are fed into bit-plane encoders, which compress the DBX and DBP values to a bit-stream following Table I. Most of these encodings are applied independently per DBX symbol. However, the first can be used to jointly encode multiple consecutive bit-planes at once, if they are all zero. This is where the correlation of neighboring values is best exploited. Note also the importance of the XOR-ing step in order to map two's complement negative values close to zero also to words consisting mostly of zero-bits. The proposed compression method can be applied to integers of various

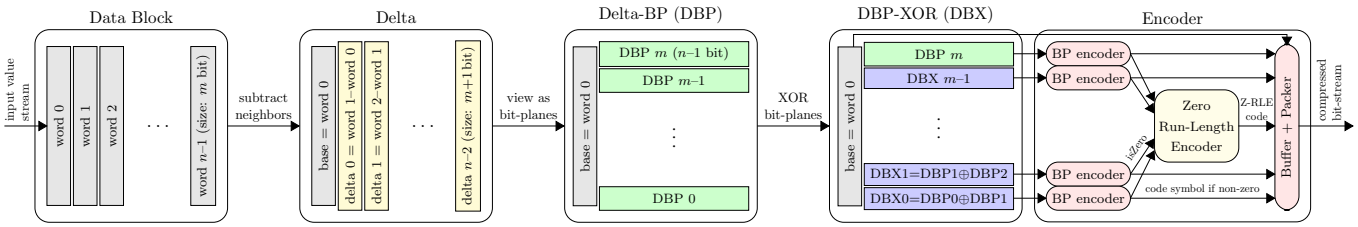

Fig. 1: Overview of the processing steps to apply bit-plane compression.

TABLE I. BIT-PLANE SYMBOL ENCODER

DBX Pattern	Length [bit]	Code (binary)
0 (run length 2 to m+1)	$2 + \log_2 m$	001 & to_bin(runLength-2)
0 (run length 1)	3	01
All-1	5	00000
DBX!=0 && DBP=0	5	00001
Two consecutive 1s	$5 + \log_2 m$	00010 & to_bin(posOfFirstOne)
Single 1	$5 + \log_2 m$	00011 & to_bin(posOfOne)
Uncompressed 1	$1 + m$	1 & to_bin(DBX word)

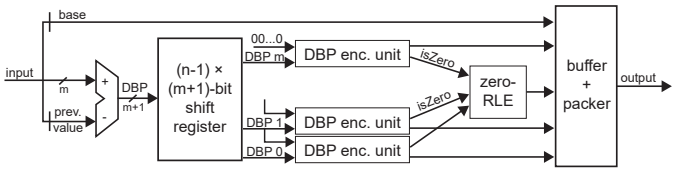

Fig. 3: Block diagram of the bit-plane encoder shown in Fig. 1.

word widths. However, it also works with floating-point words which are known to be notoriously hard to compress. While the DBX symbols corresponding to the fraction bits are almost equiprobable '1' or '0', those for the exponent and sign bits are often all-zero and thus compressible. In case of floating-point values, the bit-widths in Fig. 1 change slightly, because no additional bit is needed after the subtraction to generate the Delta values.

C. Hardware Suitability

The proposed algorithm is very hardware friendly: no codebook needs to be stored, just a few data words need to be kept in memory. From the overview (cf. Fig. 2), the Zero-RLE mostly consists of a counter and the non-zero check is also negligible in size. The buffer and packer assembles the bit-stream and needs very little logic and a few bits of storage to pack the resulting bit-stream into words. The last remaining unit, the bit-plane encoder, is shown in Fig. 3. In terms of registers, only the *base* value (m bit), the previous value to build the differences (m bit), and a $(n-1) \times (m+1)$ bit shift register are needed (with e.g. $n = m = 16$ a total of < 300 bit). Only very little logic is required as well: a single subtractor, a simple zero-RLE encoder, and the DBP encoder unit realizing the mapping in Table I.

Also the logic operations are very regular and fairly low-cost in terms of size and energy. The resulting compression reduces the energy spent on interfaces to DRAM, on inter-chip or back-plane communication—the corresponding standards specify very efficient power-down modes [32], [33]—as well as potentially saving DRAM refresh cycles for the saved memory area [1], and providing an alternative to increasing the bandwidth of such interfaces, which would imply more expensive packages, circuit boards, and additional on-chip circuits (e.g. PLLs, on-chip termination, etc.) [32], [33].

IV. RESULTS

A. Experimental Setup

Where not otherwise stated, we perform our experiments on AlexNet and are using images from the ILSVRC validation set. All models we used were pre-trained and downloaded from the PyTorch/Torchvision data repository. Some of the experiments are performed with fixed-point data types (default: 16-bit fixed-point). For these, the feature maps were normalized to exploit the full range, i.e. the worst-case scenario from a compression point of view. All the feature maps were extracted after the ReLU activations.

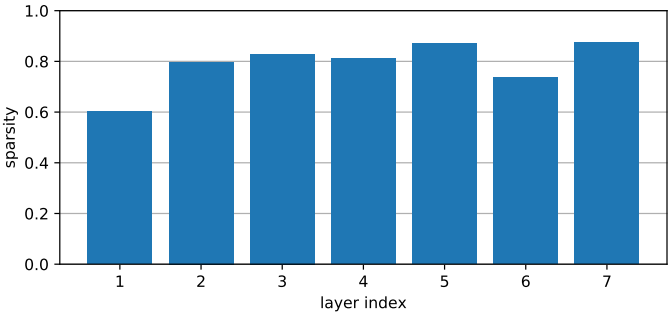

Fig. 4: Feature map sparsity after activation for each layer in AlexNet.

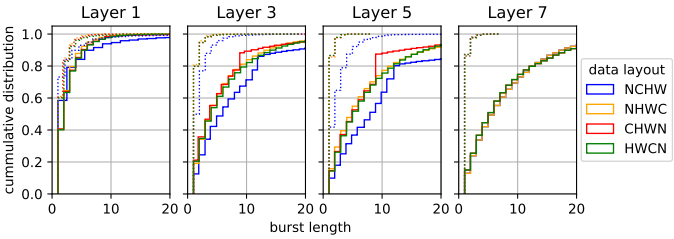

Fig. 5: Cumulative probability distribution of zero (solid) and non-zero (dotted) burst lengths.

B. Sparsity, Activation Histogram & Data Layout

Neural networks are known to have sparse feature maps after applying a ReLU activation layer, which can be applied on-the-fly after the convolution layer and possibly batch normalization. However, it varies significantly for different layers within the network as well as for different CNNs. Sparsity is a key aspect when compressing feature maps, and we analyze it in Fig. 4.

The sparse values are not independently distributed but rather occur in bursts when the 4D data tensor is laid out in one of the obvious formats. The most commonly used formats are NCHW and NHWC, which are those supported by most frameworks and the widely used Nvidia cuDNN backend. NCHW is the preferred format for cuDNN and the default memory layout and means that neighboring values in horizontal direction are stored next to each other in memory before the vertical, channel, and batch dimensions. NHWC is the default format of TensorFlow and has long before been used in compute vision and has the advantage of simple non-strided computation of inner products in channel (i.e. feature map) dimension. Further reasonable options which we include in our analysis are CHWN and HWCN, although most use-cases with hardware acceleration are targeting real-time low-latency inference and are thus operating with a batch size of 1. We analyze the distribution of the length of zero bursts for the these four data layouts at various depths within the network in Fig. 5.

The results clearly show that having the spatial dimensions (H, W) in next to each other in the data stream provide the longest zero bursts (lowest cumulative distribution curve) and

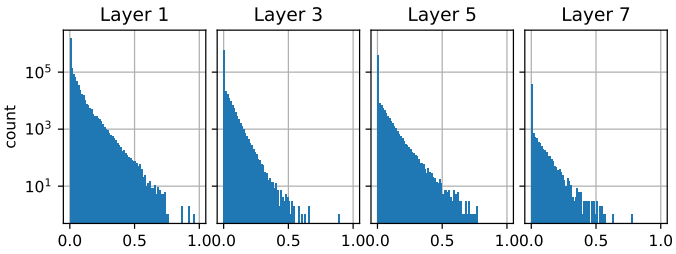

Fig. 6: Histogram of activation values at various depths within the network. Note the logarithmic vertical axis.

TABLE II. COMPRESSION RATIO USING ZVC AND ZERO-RLE FOR VARIOUS MAXIMUM ZERO BURST LENGTHS

wordwidth	ZVC	Zero-RLE max. zero burst length					
		2^1	2^2	2^3	2^4	2^5	2^6
8	2.52	2.48	2.56	2.62	**2.63**	2.59	2.53
16	3.00	2.96	3.02	3.06	**3.07**	3.04	3.00
32	3.30	3.28	3.32	3.34	**3.35**	3.33	3.31

thus the better compressibility than the other formats. This is also aligned with intuition: feature maps values mark the presence of certain features and can be expected to be smooth. Inspection the feature maps of CNNs is commonly known to show that they behave like 'heat maps' marking the presence of certain geometric features nearby. Based on these results we perform all the following evaluations based on the NCHW data layout. Not also that the burst length of non-zero values is mostly very short, such that there is limited gain in applying RLE also for the one-bits.

To compress further beyond exploiting the sparsity, the data has to remain compressible. This is definitely the case as can be seen when looking at histograms of the activation distributions as shown in Fig. 6 and a strong indication that additional compression of the non-zero data is possible.

C. Selecting Parameters

The proposed method has two parameters: the maximum length of a zero sequence that can be encoded with a single code symbol of the Zero-RLE, and the BPC block size (n, number of non-zero word encoded jointly).

Max. Zero Burst Length: We first analyze the effect of varying the maximum zero burst length for Zero-RLE on the compression ratio without for various data wordwidths in Table II. The optimal value is arguably the same for our proposed method, since a constant offset in compressing the non-zero values does not affect the optimal choice of this parameter (just like to wordwidth has no effect on it). The results also serve as a baseline for Zero-RLE and ZVC. It is worth noting that ZVC corresponds to Zero-RLE with a max. burst length of 1, yet breaks the trend shown in Table II. This is due to an inefficiency of Zero-RLE in this corner: for a zero burst length of 1, ZVC requires 1 bit whereas Zero-RLE with a max. burst length of 2 takes 2 bit. For a zero burst of length 2, ZVC encode 2 symbols of 1 bit each and Zero-RLE takes 2 bit as well. ZVC thus always performs at least as well for such a short max. burst length.

BPC Block Size: We analyze the effect of the BPC block size parameter in Fig. 7 at various depths within the network. The best compression ratio is achieved with a block size of 16

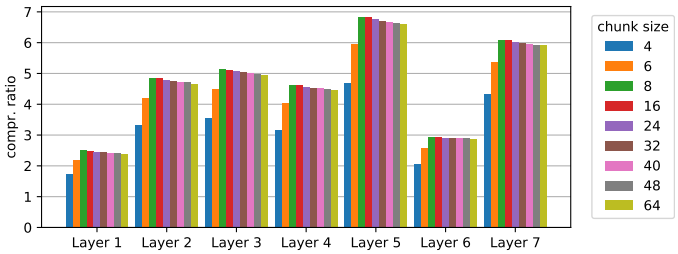

Fig. 7: Analysis of the compression ratios of various layers' outputs for several BPC block sizes and with 16-bit fixed-point values.

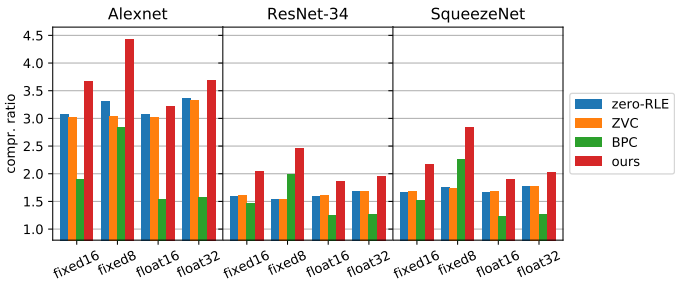

Fig. 8: Evaluation of the compression ratio of all feature maps of AlexNet, ResNet-34, and SqueezeNet for various compression methods and data types.

across all the layers. A block size of 8 might also be considered to minimize the resources of the (de-)compression hardware block at a small drop in compression ratio.

D. Total Compression Factor

We analyze the total compression factor of all feature maps of AlexNet, ResNet-34, and SqueezeNet in Fig. 8. For AlexNet, we can notice the high compression ratio of around 3× already introduced by Zero-RLE and ZVC and that it is very similar for all data types. We further see that pure BPC is not suitable since it introduces too much overhead when encoding only zero-values. For ResNet-34 and SqueezeNet, the gains by exploiting only the sparsity is significantly lower at around 1.55× and 1.7×. The proposed method outperforms previous approaches clearly with compression ratios of 4.45×, 2.45×, and 2.8× (for 8-bit fixed-point), respectively.

The gains for 8-bit fixed-point data is significantly higher than for other data formats. Most input data—also CNN feature maps—carry the most important information is in the more significant bits and in case of floats in the exponent. The less significant bits appear mostly as noise to the encoder and cannot be compressed without accuracy loss, such that this behavior—a lower compression ratio for wider word widths—is expected.

V. CONCLUSION

We have presented and evaluated a novel compression method for CNN feature maps. The proposed algorithm achieves an average compression ratio of 4.4× on AlexNet (+35% over previous methods), 2.45× on ResNet-34 (+60%), and 2.8× on SqueezeNet (+65%) for 8 bit data, and thus clearly outperforms state-of-the-art, while fitting a very tight hardware resource budget with <300 bit of data and very little compute logic.

REFERENCES

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, no. December, pp. 60–88, 2017.

[2] X. Bian, S. N. Lim, and N. Zhou, "Multiscale fully convolutional network with application to industrial inspection," in *Proc. IEEE WACV*, mar 2016.

[3] L. Cavigelli, D. Bernath, M. Magno, and L. Benini, "Computationally efficient target classification in multispectral image data with Deep Neural Networks," in *Proc. SPIE Secur. + Def.*, vol. 9997, 2016.

[4] B. Wu, F. Iandola, P. H. Jin, K. Keutzer, and U. C. Berkeley, "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving," in *Proc. IEEE CVPRW*, 2016, pp. 129–137.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proc. IEEE ICCV*, dec 2015, pp. 1026–1034.

[6] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *IEEE TCSVT*, vol. 27, no. 11, pp. 2461–2475, nov 2017.

[7] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *Proc. ACM GLSVLSI*. ACM Press, 2015, pp. 199–204.

[8] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," in *Proc. ACM/IEEE ISCA*, 2016.

[9] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," in *Proc. ACM/IEEE ISCA*, 2017, pp. 27–40.

[10] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. IEEE/ACM MICRO*, oct 2016.

[11] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *IEEE Trans. Neural Networks Learn. Syst.*, jun 2018.

[12] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proc. ACM/IEEE ISCA*, 2016, pp. 367–379.

[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proc. ACM/IEEE ISCA*, 2016, pp. 243–254.

[14] L. Cavigelli and L. Benini, "CBinfer: Exploiting Frame-to-Frame Locality for Faster Convolutional Network Inference on Video Streams," pp. 1–13, 2018.

[15] L. Cavigelli, M. Magno, and L. Benini, "Accelerating Real-Time Embedded Scene Labeling with Convolutional Networks," in *Proc. ACM/IEEE DAC*, 2015.

[16] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Hyperdrive: A Systolically Scalable Binary-Weight CNN Inference Engine for mW IoT End-Nodes," in *Proc. IEEE ISVLSI*, 2018.

[17] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Adv. NIPS*, 2015.

[18] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *Proc. IEEE ISVLSI*, 2016, pp. 236–241.

[19] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *Proc. ICLR*, 2017.

[20] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration," *IEEE TCAD*, 2017.

[21] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," *JMLR*, vol. 37, 2015.

[22] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool, "Soft-to-hard vector quantization for end-to-end learning compressible representations," in *Adv. Neural Inf. Process. Syst.*, 2017.

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

[24] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *Proc. IEEE HPCA*, 2018, pp. 78–91.

[25] D. A. Gudovskiy, A. Hodgkinson, and L. Rigazio, "DNN Feature Map Compression using Learned Representation over GF(2)," *arXiv:1808.05285*, no. 2, 2018.

[26] Y. Wang, C. Xu, C. Xu, and D. Tao, "Beyond Filters: Compact Feature Map for Portable Deep Model," in *Proc. ICML*, vol. 70, 2017, pp. 3703–3711.

[27] T. A. Welch, "A Technique for High-Performance Data Compression," *Computer (Long. Beach. Calif).*, vol. 17, no. 6, pp. 8–19, jun 1984.

[28] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[29] M. B. Lin, J. F. Lee, and G. E. Jan, "A lossless data compression and decompression algorithm and its hardware architecture," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 9, pp. 925–936, 2006.

[30] X. Zhou, Y. Ito, and K. Nakano, "An efficient implementation of LZW decompression in the FPGA," *Proc. IEEE IPDPS*, pp. 599–607, 2016.

[31] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures," in *Proc. IEEE ISCA*, 2016, pp. 329–340.

[32] "Low Power Double Data Rate 4 (LPDDR4)," JEDEC Solid State Technology Association, Tech. Rep. August, 2014.

[33] "Graphics Double Data Rate (GDDR5) SGRAM," JEDEC Solid State Technology Association, Tech. Rep. Feburary, 2016.