Cyclic Sparsely Connected Architectures for Compact Deep Convolutional Neural Networks

Morteza Hosseini, *Member, IEEE*, Nitheesh Kumar Manjunath, Bharat Prakash, Arnab Mazumder, *Graduate Student Member, IEEE*, Vandana Chandrareddy, Houman Homayoun, and Tinoosh Mohsenin

***Abstract*— In deep convolutional neural networks (DCNNs), model size and computation complexity are two important factors governing throughput and energy efficiency when deployed to hardware for inference. Recent works on compact DCNNs as well as pruning methods are effective, yet with drawbacks. For instance, more than half the size of all MobileNet models lies in their last two layers, mainly because compact separable convolution (CONV) layers are not applicable to their last fully connected (FC) layers. Also, in pruning methods, the compression is gained at the expense of irregularity in the DCNN architecture, which necessitates additional indexing memory to address nonzero weights, thereby increasing memory footprint, decompression delays, and energy consumption. In this article, we propose cyclic sparsely connected (CSC) architectures, with memory/computation complexity of $\mathcal{O}(N \log N)$, where N is the number of nodes/channels given a DCNN layer that, contrary to compact depthwise separable layers, can be used as an overlay for both FC and CONV layers of $\mathcal{O}(N^2)$. Also, contrary to pruning methods, CSC architectures are structurally sparse and require no indexing due to their cyclic nature. We show that both standard convolution and depthwise convolution layers are special cases of the CSC layers, whose mathematical function, along with FC layers, can be unified into one single formulation and whose hardware implementation can be carried out under one arithmetic logic component. We examine the efficacy of the CSC architectures for compression of LeNet, AlexNet, and MobileNet DCNNs with precision ranging from 2 to 32 bits. More specifically, we surge upon the compact 8-bit quantized 0.5 MobileNet V1 and show that by compressing its last two layers with CSC architectures, the model is compressed by ~1.5× with a size of only 873 kB and little accuracy loss. Finally, we design a configurable hardware that implements all types of DCNN layers including FC, CONV, depthwise, CSC-FC, and CSC-CONV indistinguishably within a unified pipeline. We implement the hardware on a tiny Xilinx field-programmable gate array (FPGA) for total on-chip processing of the compressed MobileNet that, compared to the related work, has the highest Inference/J while utilizing the smallest FPGA.**

Manuscript received December 28, 2020; revised April 27, 2021 and July 14, 2021; accepted August 11, 2021. Date of publication September 15, 2021; date of current version October 6, 2021. *(Corresponding author: Morteza Hosseini.)*

Morteza Hosseini, Nitheesh Kumar Manjunath, Bharat Prakash, Arnab Mazumder, Vandana Chandrareddy, and Tinoosh Mohsenin are with the Department of Computer Science and Electrical Engineering, University of Maryland at Baltimore County, Baltimore, MD 21250 USA (e-mail: hs10@umbc.edu).

Houman Homayoun is with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3110250.

Digital Object Identifier 10.1109/TVLSI.2021.3110250

***Index Terms*— Application-specific integrated circuit (ASIC), compressed deep convolutional neural networks (DCNNs), cyclic sparsely connected (CSC) architectures, field-programmable gate array (FPGA), neural network accelerator.**

I. Introduction

Machine learning has reached a point where it can surpass human-level accuracy in tasks, such as speech recognition [1], [2] and computer vision [3]. In most recent systems, many of these tasks are implemented using artificial neural networks (ANNs) that are basically nonlinear mathematical functions, inspired by biological neural networks, which takes in an input data (e.g., an RGB image) and calculates an output result (e.g., an evaluated object in an image). Generally, ANNs encompass a wide range of types including deep neural networks (DNNs),[1] deep convolutional neural networks (DCNNs), recurrent neural networks, and their combinations and variations.

DNNs were originally introduced as universal approximators made with nonlinear multilayer feedforward networks [4]. DCNNs are a variation of DNNs that employ convolution/correlation functions at their core to identify temporal/spatial correlations within the input data. DCNNs are made of a few layers formed in a directed acyclic graph (DAG) [5]. Typically, the layers are of type convolution (CONV), FC, nonlinear activation function, batch normalization, and max pool. Computation in a DCNN is attributed to an extensive amount of multiply–accumulate (MAC) operations, and the model size is attributed to the number of parameters, both of which are commonly dominated by CONV and FC layers. Modern DCNNs trained on elaborate datasets suffer from large model size and high computation that makes their deployment to resource-bound hardware platforms, such as embedded or assistive devices challenging. Therefore, many researching efforts are taken to explore compressing techniques that reduce the computation and the model size while maintaining accuracy, aiming for DCNN optimal architecture exploration and efficient deployment to resource-constrained platforms ranging from general-purpose processing devices [1], [3] to application-specific integrated circuits (ASICs) [6]–[9] and field-programmable gate arrays (FPGAs) [10]–[14].

In this article, we review the state-of-the-art compressing methods for DCNNs, and under the category of compact network design, we propose cyclic sparsely connected (CSC) architectures as a factorized overlay for DCNN layers that, compared to standard DCNN layers, can potentially reduce

[1]Following notation in [1], we refer to a multilayer network as a DNN if all layers are of type fully connected (FC), otherwise as a DCNN if CONV layers are included.

both computation and size of a layer from $\mathcal{O}(N^2)$ down to $\mathcal{O}(N \log N)$, where N is the number of channels/nodes given a layer. The CSC architectures are composed of a few structurally sparse layers cascaded, which results in full connectivity between the input and output nodes of the cascaded layers. We formulate CSC architectures for both their formation and their operation when being used as a DCNN layer and evaluate their compression and their impact on accuracy. Furthermore, we show that both standard CONV layers as well as depthwise separable CONV layers [3] can be considered as special cases of CSC layers. Finally, we design a hardware to implement DCNNs from a CSC perspective and show that the hardware can effectively implement CSC DCNNs as well as standard DCNNs and depthwise separable DCNNs. The main contributions of this article include the following.

1) Propose two types of CSC architectures with reduced computational complexity and two schemes for CSC-CONV layers as substitutes for FC and CONV layers.
2) Train LeNet300-100, AlexNet, and MobileNet-224 and −192 using CSC architectures, with compression of 19×, 7.3×, and 1.5×, respectively, compared to their baselines within a margin of 1.5% accuracy loss.
3) Propose a method of double compression by adopting CSC architectures in tandem with extreme quantization.
4) Propose a scalable hardware, configured with 192 MAC units and 8-bit dataflow, implemented on a tiny Artix-7 FPGA that takes total advantage of on-chip block RAMs (BRAMs) to deploy the compressed 8-bit 0.5 MobileNet-192 V1 for inference.

The rest of this article is organized as follows. Section II underlines the motivations underlying our proposed method. Section III overviews the related works. Section IV introduces and formulates the CSC architectures from their foundation to their application. In Section V, CSC architectures are formulated in terms of computation when being adopted by either FC or CONV layers for which two schemes are proposed. In Section VI, the schemes are applied to popular DCNNs and are compared against the related compressing methods. Section VII proposes a hardware accelerator that implements DCNNs with or without compression with CSC architectures. Section VIII compares the characteristics of the hardware implementation to the state of the art. Finally, Section IX presents the conclusion.

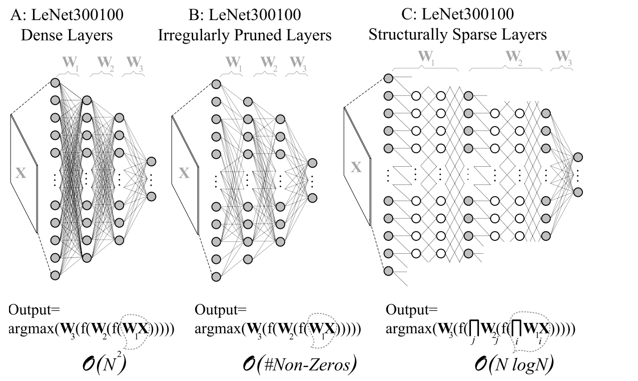

A: LeNet300100
Dense Layers
B: LeNet300100
Irregularly Pruned Layers
C: LeNet300100
Structurally Sparse Layers
W_1 W_2 W_3
X
Output= argmax($\mathbf{W}_3$(f($\mathbf{W}_2$(f($\mathbf{W}_1\mathbf{X}$)))))
$\mathcal{O}(N^2)$
Output= argmax($\mathbf{W}_3$(f($\mathbf{W}_2$(f($\mathbf{W}_1\mathbf{X}$)))))
$\mathcal{O}$(#Non-Zeros)
Output= argmax($\mathbf{W}_3$(f($\prod_j \mathbf{W}_{2j}$(f($\prod_i \mathbf{W}_{1i}\mathbf{X}$)))))
$\mathcal{O}(N \log N)$

Fig. 1. LeNet-300-100: (a) baseline DNN with dense layers/operations, (b) pruned model with sparse layers that requires indexing and a decompression mechanism to perform operations between nonzero values in W and values from X fetched from memory based on the index of nonzero values in W, and (c) compressed with a structural CSC model that requires no indexing.

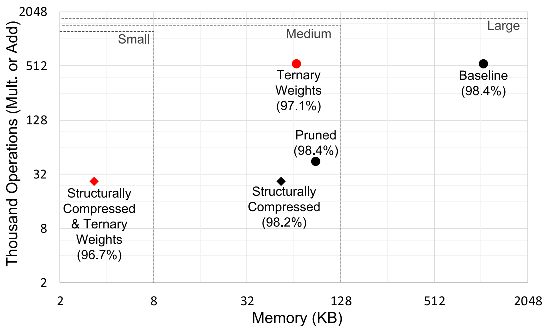


Fig. 2. Model size versus operation count of a 32-bit LeNet-300-100 and four of its compressed variants, including a 2-bit model, a 32-bit pruned model [15], a 32-bit structurally compressed CSC, and its 2-bit model, compressed by 16×, 12×, 19×, and 304×, respectively, highlighting the advantage of structural sparsity in tandem with extreme quantization.

II. Motivation

A. Pruning Versus Structured Sparsity

Despite their effectiveness, fine-grained pruning methods result in the irregularity of the pattern of nonzero weights in the pruned model that necessitates additional indexing. Thus, their compressed model has more parameters than the sheer amount of nonzero weights, and their implementation is deteriorated by the model decompression. As an example, our experiments on the NVIDIA TX2 GPU at clock rate 1.3 GHz indicate that a matrix–vector multiplication using a matrix of size 16 384 × 16 384 can be implemented by the NVIDIA CuBLAS library with performance ∼11.4 GOPS, whereas the implementation of the same matrix with 95% zero values compressed with compressed sparse row (CSR) storage format [16] can be executed using the CuSPARSE library with performance ∼3.9 GOPS that merely operates over the 5% of nonzero values. Despite the fact that CuSPARSE can implement the latter problem 6.8× faster than the CuBLAS, its advantage is gained by the existence of 20× less number of operations, yet its degraded performance is caused by the decompressing algorithm applied to the matrix compressed by the CSR format. In structured sparsity, on the other hand, indexing nonzero values is eliminated and the implementation can be handled as high performance as dense matrices. Fig. 1 reflects this motivation by illustrating the computation complexity of a LeNet-300-100 DNN compressed with either pruning or structurally sparse method used in this work.

B. Low Bitwidth Neural Networks

When quantization and pruning methods are used in tandem, unlike the DCNN weights and activations, the extra indexing memory imposed by the pruning method is not quantizable, and therefore, the extra indexing memory might be intolerable in resource-bound platforms that employ low bitwidth such as binary [22] or ternary [23] weight neural networks. Pruning such neural networks can result in a larger model size compared to their uncompressed models. For instance, if a ternary weight 16 384 × 16 384 matrix with 90% zero values is compressed with a COO (coordinate) format [16], it requires 97 MB of storage (dedicating 1 and 28 bits to the nonzero value and its index, respectively), whereas its uncompressed model requires 67 MB (dedicating 2 bits per matrix weight). Thus, our second motivation is to develop and employ structurally compact models that require no indexing

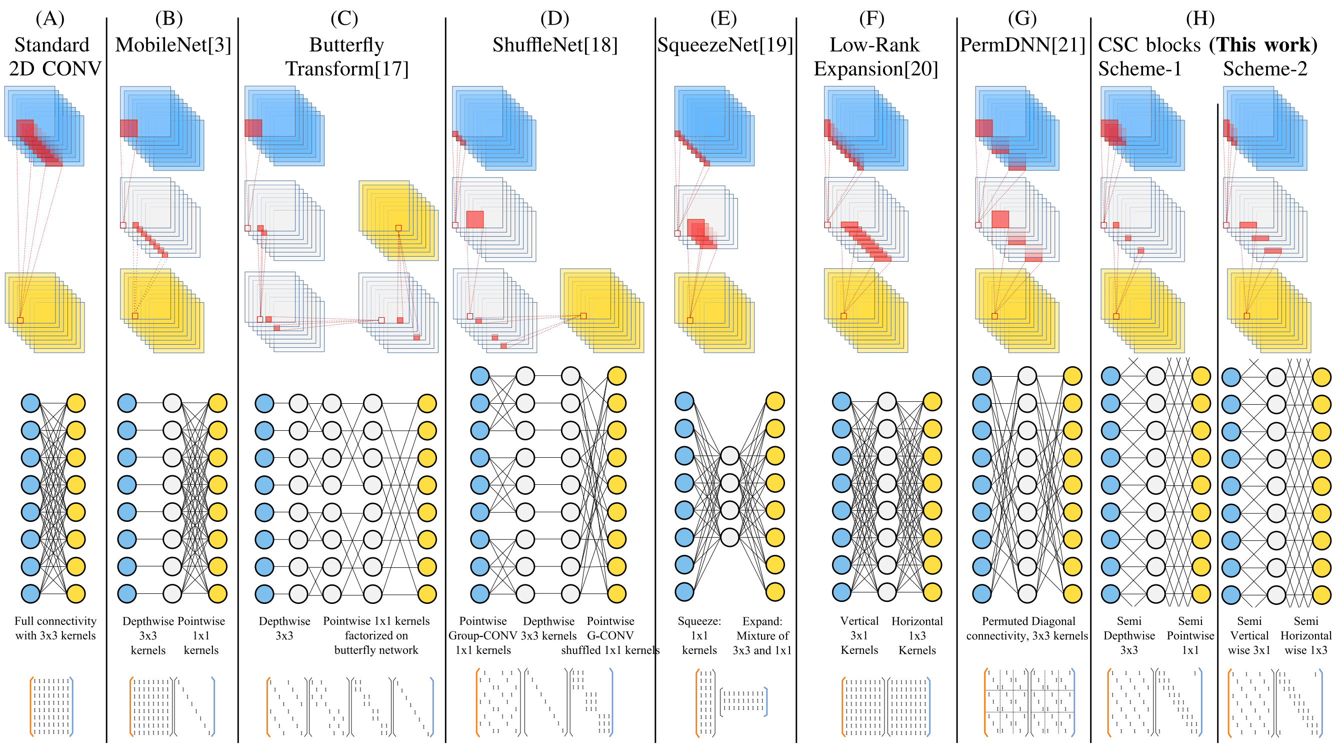


Fig. 3. Key ideas in compact schemes of novel works illustrated with a typical multilayer architecture and equivalent structured graphs whose connectivity is denoted with biadjacency matrices. Blue, gray, and yellow channels/nodes represent input, intermediate, and output feature maps, respectively. All these works share two things in commons: 1) the product of all biadjacency matrices is either an all-one or an all-C matrix where C is the #paths connecting arbitrary I/O nodes and 2) the kernels are laid on the edges such that any path connecting any I/O channels/nodes composes a $K \times K$ kernel (PermDNN excepted). With the two common characteristics, all these multilayer architectures with linear activations are equivalent to a standard 2-D CONV with $K \times K$ kernels.

for low precision neural networks. Fig. 2 plots the memory versus the number of operations for the same DNN with four of its compressed variants, highlighting the significant model size compression gained by combining structured sparsity and extreme quantization.

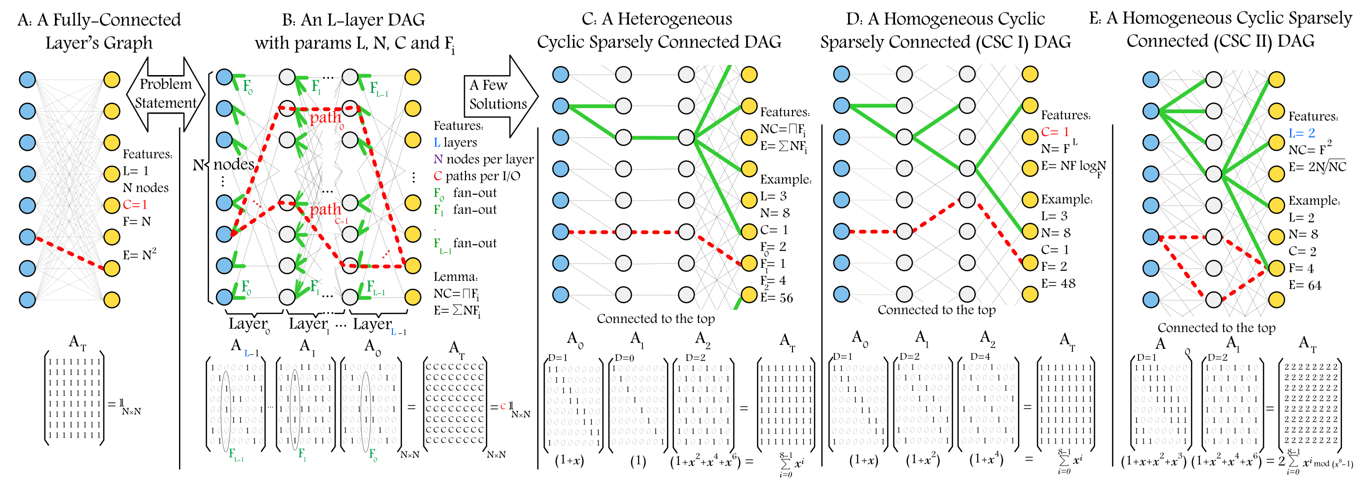
III. Related Work

General compressing methods for DCNNs include the following.

1) Quantization: Although quantization methods do not reduce the number of operations, they can reduce the DCNN model and the computation cost by altering floating-point to fixed-point operations that use simple circuitry in hardware. DoReFaNet [24] and QKeras [25] are frameworks that allow quantizing both weights and feature maps (fmaps) to any arbitrary level. The Google TensorFlow lite (TFlite) allows quantizing DCNNs to 8-bit TFlite models with negligible accuracy drop, preparing them for deployment to the efficient fixed-point pipeline of general-purpose CPUs.

2) Pruning: Pruning methods [15], [26] effectively reduce the number of nonzero parameters and consequently the number of MAC operations by identifying and removing weak weights in the DCNN and fine-tuning the strong weights. This method is also referred to as fine-grained pruning, the main drawback of which is the irregular pattern of nonzero weights in the DCNN that necessitates a compressing format such as COO and a decompression algorithm. Coarse-grained (also known as structured) pruning methods [27]–[30], on the other hand, prune DCNNs more aggressively on structured levels, such as vector, kernel, filter, group, or layer basis. Coarse-grained pruning methods can minimize the extra storage imposed by indexing as opposed to fine-grained methods, and the groups of nonzero entries can together, rather individually, be located with single indicators, thus minimizing indexing.

3) Compact Models: Contrary to pruning methods that follow a train-prune routine, compact models can be considered as a prune-train routine in which the model is forced to be trained on an already-sparsified basis. In Fig. 3, a few popular compact models from the literature are illustrated with their equivalent graphs. Every input node from a graph represents a 2-D channel, and each edge is weighted with a 2-D kernel, the connection of which corresponds to a 2-D convolution operation. Every output node is a 2-D channel that is resulted by summing over all of its connecting kernels operated on their connected inputs. Below each graph, the connectivity matrices from left to right represent the topology of the graph from right to left. Compact models, such as low-rank expansion [20], MobileNet [3], ShuffleNet [18], SqueezNet [19], PermDNN [21], PermCNN [9], and Butterfly Transform [17], as shown in Fig. 3 introduce alternate presparsified layers with a common intuition: in all of them, each model proposes a predefined factorization that can be equated to a standard DCNN layer. It is their factorization form that allows compact representation and reduced operations. In other words, if a standard layer in a DCNN could be factorized into sparse layers, then the factorized model could result in a smaller representation with less computation. Luckily, training DCNNs does not result in a unique solution. Therefore, one can define a sparse factorized basis in advance and, on which, enforce the DCNN to be trained. In MobileNet as an example, it is shown that standard 2-D convolution layers can be replaced with a depthwise and a pointwise convolution layer that jointly together constitutes a separable layer, the characteristics of which is similar to the standard CONV layer, yet with



A: A Fully-Connected Layer's Graph
B: An L-layer DAG with params L, N, C and F_i
C: A Heterogeneous Cyclic Sparsely Connected DAG
D: A Homogeneous Cyclic Sparsely Connected (CSC I) DAG
E: A Homogeneous Cyclic Sparsely Connected (CSC II) DAG
Problem Statement
A Few Solutions
Features: L= 1 N nodes C=1 F= N E= N²
Features: L layers N nodes per layer C paths per I/O F_0 fan-out F_1 fan-out F_L-1 fan-out
Lemma: NC=∏F_i E=∑NF_i
Features: NC=∏F_i E=∑NF_i
Example: L= 3 N= 8 C= 1 F_0= 2 F_1= 1 F_2= 4 E= 56
Features: C= 1 N= F^L E= NF log_F N
Example: L= 3 N= 8 C= 1 F= 2 E= 48
Features: L= 2 NC= F² E= 2N√NC
Example: L= 2 N= 8 C= 2 F= 4 E= 64
Connected to the top
path_0
path_C-1
N nodes
Layer_0 Layer_1 Layer_L-1
A_T A_0 A_1 A_2 A_L-1
(1+x) (1) (1+x²+x⁴+x⁶)
(1+x) (1+x²) (1+x⁴)
(1+x+x²+x³) (1+x²+x⁴+x⁶)

Fig. 4. Exploring an architecture that approximates a weighted dense layer into factorization of weighted sparse layers. (a) FC graph with size N, (b) problem statement highlighting a DAG with L layers, N nodes per layer, equally fanned out layers, and equally connected I/O nodes with C paths, a few solutions of which can be (c) heterogeneous CSC graph that has differently fanned-out layers, (d) homogeneous CSCI graph where $C = 1$, and (e) homogeneous CSCII graph where #layers (L) is 2. For all of the DAGs, biadjacency matrices at the bottom of the figure highlight the connectivity of the nodes, the product of which reveals #paths for all I/O nodes. Generator polynomials underline the formation of each graph.

approximately K^2 times less parameters and less computation if $K \times K$ kernels have a high number of channels [3].

IV. Cyclic Sparse Connections

We first introduce and formulate the CSC architectures from a graph theory perspective. Then, in Section V, we use these architectures for compact DCNN design. Furthermore, in Section VII, we take advantage of CSC architectures in design of a high-bandwidth router in our DCNN accelerator hardware. The novel routing network can be categorized as a high-bandwidth communication network for parallel system interconnections.

Throughout this work, we use italic lower case letter x to represent generator polynomials (e.g., $p(x) = 1 + x + x^2$), italic capital letters (e.g., N) for integer values, bold upper case letters for matrices (e.g., $\mathbf{W}$) and vectors (e.g., $\mathbf{X}$), and calligraphic uppercase letters for matrix of matrices to represent tensors (e.g., $\mathbb{F}$). We use italic lower case letters in brackets for the elements of a matrix or a vector (e.g., $\mathbf{W}[i, j]$ or $\mathbf{X}[i]$) and parenthesis for matrix elements of a tensor [e.g., $\mathbb{F}(h, w)$]. Thus, the parameters of a 3-D tensor such as a feature map data or a 4-D tensor such as a filter set can be represented with ensemble of brackets and parenthesis [e.g., $\mathbb{X}[h, w](i)$ or $\mathbb{F}[h, w](i, j)$].

A. Problem Statement and Formulation

Concluded from the related works [3], [17]–[21] and more specifically inspired by the structures of Butterfly networks in fast Fourier transform (FFT) [31], we observe that in their graphs, the degree (number of connected edges) of every node/channel within the same layer is equal and there exist an equal number of paths connecting each input node to every output node. The objective of this section is hence to formulate a DAG composed of a few layers where all input nodes are connected via an equal number of paths to all output nodes of the graph. The graph is composed of an input layer, $L - 1$ layers in between referred to as factorized layers, and an output layer, each layer of size N. We denote the first (input) and the last (output) layers by capitalizing their first letters. We call it a homogeneous graph if all nodes in the graph are equally fanned-out and equally fanned-in with hyperparameter F or a heterogeneous graph if nodes at layer l are equally fanned-out with individual hyperparameter F_l. The total number of edges E in this graph can be counted as follows:

$$E = N \sum_{l=0}^{L-1} F_l. \tag{1}$$

To build upon our problem statement, we define a connectivity metric C that defines the number of paths that connect each arbitrary input node to every arbitrary output node from the graph. Fig. 4(a)–(e) shows an FC graph, a DAG, and its parameters intended to approximate a weighted dense layer into factorization of weighted sparse layers, a heterogeneous graph, a homogeneous graph with $C = 1$, and a homogeneous graph with $C = 2$, respectively. Thus, for a heterogeneous graph

$$\prod_{l=0}^{L-1} F_l = NC. \tag{2}$$

As a result, in homogeneous graphs, $E = NFL$ and $F^L = NC$. The objective of homogeneity is to provide a basis where every arbitrary node in the graph is equally exploited, every arbitrary pair of nodes from input and output layers is equally connected, the information flux through the factorized layers is fairly equal, and the rank of the transforming weighted matrix can remain full. Given $F_l = F$ for all layers, combining (1) and (2), a logarithmic relationship between the number of edges and size of layers is obtained for a homogeneous graph as follows:

$$E_{\text{homo.}} = NF \log_F (NC) \tag{3}$$

that reflects the compression it provides compared to an FC graph with N^2 synapses. As an example, given $F = 2$ and $C = 1$, (2) and (3) infer that $L = \log_2 N$ and $E = 2N\log_2 N$, which is the case in radix-2 butterfly networks for FFT [31]. As a comparison, the graph in the Fig. 3(c), with the leftmost layer excluded, represents a radix-2 ($F = 2$) butterfly network for an eight-point discrete FFT ($N = 8$). The graph has three layers ($= \log_2 8$) and 48 synapses ($= 2\times 8\log_2 8$) and clearly shows one and only one path ($C = 1$) between arbitrary pair of nodes from its input–output layers.

1) Biadjacency Matrices: For every layer in the graph, we define a biadjacency matrix $\mathbf{A}$ as the connectivity matrix whose height and width is equal to the input and output size of the layer, respectively, whose elements $\mathbf{A}(i, j)$ indicate the number of edges that connect the input node i to the output node j. Therefore, NF out of N^2 elements of the biadjacency matrix of every factorized layer in our problem statement are 1 and the rest are 0.

2) Input–Output Adjustment: To replace an FC graph that has input size N_I and output size N_O with a homogeneous graph that has a different N, we tile and truncate only the input and output layers of the replacing graph to match them to those of the baseline graph. To adjust the input layer to an input vector of size N_I, we remove rows from the bottom of the biadjacency matrix if $N_I < N$ (truncation) or we recursively copy from the first consecutive rows of the biadjacency matrix and add them to its bottom if $N_I > N$ (tiling) until its number of rows is equal to N_I. For the output layer to be adjusted to an output of size N_O, we manipulate the columns of the last biadjacency matrix similarly. By doing so, the connectivity between the adjusted input and output layers still remains C. For a homogeneous graph with parameters N, F, L, C, and adjusted I/O sizes of N_I and N_O, the number of edges are

$$E_{\text{adj}} = NF(L-2) + N_I F + N_O F. \tag{4}$$

B. Solution to the Problem Statement

There is no unique solution for graphs that meet the problem statement. We show that circulant matrices generated from generator polynomials as in cyclic error-correcting codes give a set of solutions for the biadjacency matrices of the layers that satisfy all the requisites in our problem statement. From here on, we call these graphs CSC architectures. Suppose that the biadjacency matrix $\mathbf{A}_l$ $(0 \le l \le L-1)$ of every factorized layer l in our CSC graph has a generator polynomial $p_l(x)$ that has F terms, which generates a cyclic biadjacency matrix of block length N. It can be shown that the product of the L biadjacency matrices attributed to the L layers is a matrix $\mathbf{A}_T$, where $\mathbf{A}_T(i, j)$ represents the number of paths between the input node i and output node j of the CSC graph. In the problem statement, the connectivity should be equal to C for all arbitrary pairs of input and output nodes and thus should $\mathbf{A}_T = C\mathbf{J}$, where $\mathbf{J}$ is an $N \times N$ all-one matrix. The all-C matrix $\mathbf{A}_T$ can be attributed to another generator polynomial $p_T(x) = C\sum_{i=0}^{N-1} x^i$. The generator polynomial constructs a cyclic matrix as follows: the first row of the matrix corresponds to the coefficients of the polynomial–represented as big-endian in this article—and then, every next row is a cyclic right shift of its previous row. The cyclic biadjacency matrices in this article are not ideals as in ideal cyclic codes and might have nondistinctive rows. We provide two different factorization sets of $p_T(x)$.

1) CSCI: Connectivity Equal to One: If $C = 1$ and F, L, and N are such that $N = F^L$, then $p_T(x) = \sum_{i=0}^{N-1} x^i$ can be factorized as follows:

$$\begin{cases} \displaystyle\sum_{i=0}^{N-1} x^i = \prod_{l=0}^{L-1} p_l(x) \\ \displaystyle p_l(x) = \sum_{i=0}^{F-1} x^{D_l.i}, \quad D_l = F^l. \end{cases} \tag{5}$$

By assigning $p_l(x)$ as the generator polynomial of factorized layer l, the CSC layers of the graph are completely defined. D_l is the dilation parameter that indicates the distance between elements of value 1 in the first row of the biadjacency matrix of layer l. It also represents the dilation between the fanned-out edges in its corresponding layer. For small values of F (e.g., 2 or 3) and consequently more number of layers (e.g., $\log_2 N$ or $\log_3 N$), the least number of synapses is resulted, but at the expense of elongating the graph that, by itself, will increase the memory communication during hardware implementation. Fig. 4(d) shows a CSCI graph with three layers and generator polynomial $p_l(x) = \sum_{j=0}^{2-1} x^{2^l j}$ for the layer l. We evaluate LeNet-300-100 on MNIST by replacing FC layers with CSCI.

2) CSCII: Layers Equal to Two: If $L = 2$ and F, C, and N are integers to satisfy $NC = F^2$, then $p_T(x) = C\sum_{i=0}^{N-1} x^i$ can be factorized as follows:

$$\begin{cases} \displaystyle C\sum_{i=0}^{N-1} x^i = p_0(x).p_1(x), \quad \bmod\left(x^N - 1\right) \\ \displaystyle p_0(x) = \sum_{i=0}^{F-1} x^{D_0.i}, \qquad D_0 = 1 \\ \displaystyle p_1(x) = \sum_{i=0}^{F-1} x^{D_1.i}, \qquad D_1 = \frac{F}{C}. \end{cases} \tag{6}$$

By assigning $p_0(x)$ and $p_1(x)$ to the first and second layers of the graph, respectively, the CSCII graph is defined. In CSCII graphs, $E = 2N\sqrt{NC}$ that declares compression given $C < (N/4)$ and a computation of $\mathcal{O}(N\sqrt{N})$. Fig. 4(e) shows a CSCII graph with $C = 2$ and generator polynomial $p_l(x) = \sum_{j=0}^{4-1} x^{2^l j}$ for layer l. In Section V, we will evaluate replacing FC and CONV layers AlexNet and MobileNet with CSCII architectures.

For both CSCI and CSCII, the biadjacency matrix of a layer with size N, fan-out F, and a dilation D_l is inferred as

$$\mathbf{A}_l(i, j) = \begin{cases} 1, & \text{if } (i-j+kD_l) \bmod N = 0, \quad \forall k = 0, 1, \ldots, (F-1) \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

V. CSC Architectures in DCNNs

For simplicity, we ignore the term bias in all equations in this section, assume that convolution operations are of same size, and the number of nodes/channels in I/O of the FC/CONV is all equal to N. As a result, vectors/matrices/tensors in this section are $\mathbf{Y} \in \mathbb{R}^N$, $\mathbf{W} \in \mathbb{R}^{N\times N}$, $\mathbf{X} \in \mathbb{R}^N$, $\mathbb{Y} \in \mathbb{R}^{W_\mathbb{Y}\times H_\mathbb{Y}\times N}$, $\mathbb{F} \in \mathbb{R}^{W_\mathbb{F}\times H_\mathbb{F}\times N\times N}$, and $\mathbb{X} \in \mathbb{R}^{W_\mathbb{X}\times H_\mathbb{X}\times N}$, where $W_\mathbb{Y}$, $W_\mathbb{F}$, and $W_\mathbb{X}$ as well as $H_\mathbb{Y}$, $H_\mathbb{F}$, and $H_\mathbb{X}$ represent width and height in channels of tensors $\mathbb{Y}$, $\mathbb{F}$, and $\mathbb{X}$, respectively. If layers are not equally sized, we adjust their I/O according to the note in Section IV-A2.

A. FC Layers

A standard FC layer is representable with a dense weight matrix $\mathbf{W}$ whose operation on an input vector $\mathbf{X}$ is equivalent to a matrix–vector multiplication that generates a vector $\mathbf{Y}$ such that

$$\mathbf{Y}[i] = (\mathbf{W}\mathbf{X})[i] = \sum_{j=0}^{N-1} \mathbf{W}[i, j]\mathbf{X}[j] \tag{8}$$

which is a computation of $\mathcal{O}(N^2)$. If $\mathbf{W}$ is factorizable into L matrices or, correspondingly, if a cascade of L CSC layers with linear activations and with parameters N and F are weighted to equate a factorizable FC layer, then the equivalent $\mathbf{W}_T = \prod_{l=0}^{L-1} \mathbf{W}_l$. Because of associative property of matrix–matrix multiplication, the computation of $\mathbf{W}_T\mathbf{X}$ can start from the rightmost matrix–vector multiplication and propagate to the leftmost matrix. As such, every individual operation between layer l with $\mathbf{W}_l$ and input $\mathbf{X}_l$ results in

$$\mathbf{Y}_l[i] = \sum_{j=0}^{F-1} \mathbf{W}_l[i, (i + jD_l) \bmod N]\mathbf{X}_l[(i + jD_l) \bmod N] \tag{9}$$

that inherently skips the zero values in $\mathbf{W}_l$ by taking only the nonzero values that are dilated D_l elements apart. Thus, the computation is of $\mathcal{O}(NF)$ for one single $\mathbf{W}_l\mathbf{X}$ and of $\mathcal{O}(NF\log_F(NC))$ for $\mathbf{W}_T\mathbf{X} = (\prod_{l=0}^{L-1} \mathbf{W}_l)\mathbf{X}$, which corresponds to cascade of L homogeneous weighted CSC layers with connectivity C and fan-out F operating on input vector $\mathbf{X}$. If $\mathbf{W}_l$ is rearranged in its compressed format $\hat{\mathbf{W}}_l \in \mathbb{R}^{N\times F}$, which is an $N \times F$ matrix that contains only NF nonzero entries of $\mathbf{W}_l$, where $\hat{\mathbf{W}}_l[i, j] = \mathbf{W}_l[i, (i + jD_l) \bmod N]$, then, (9) can be altered as

$$\mathbf{Y}_l[i] = \sum_{j=0}^{F-1} \hat{\mathbf{W}}_l[i, j]\mathbf{X}_l[(i + jD_l) \bmod N] \tag{10}$$

to which we refer as a cyclic dilated matrix–vector multiplication for a CSC-FC layer. For $F = N$ and $D = 1$, and given a new rearrangement of a dense $\mathbf{W}_l$ into a compressed format $\hat{\mathbf{W}}_l$, (10) corresponds to (8), revealing a novel approach to computation in standard matrix–vector multiplications using cyclic dilated matrix–vector multiplication.

B. Convolution Layers

With a little abuse of notation, we use the symbol "*" to denote the convolution operation. A 2-D convolution between a 4-D macro-matrix $\mathbb{F}$ and a 3-D macro-vector $\mathbb{X}$ results in a 3-D macro-vector $\mathbb{Y}$. Similar to (9), $\mathbb{Y} = \mathbb{F}*\mathbb{X}$ can be broken into summing over partial convolutions between aligned 2-D channels from $\mathbb{F}$ and $\mathbb{X}$, i.e., $\mathbb{Y}_l(i) = \sum_{j=0}^{N-1} \mathbb{F}_l(i, j) * \mathbb{X}_l(j)$ such that

$$\begin{aligned}(\mathbb{F}_l(i, j) &* \mathbb{X}_l(j))[u, v]\\ &= \sum_{w=0}^{W_F-1}\sum_{h=0}^{H_F-1} \mathbb{F}_l(i, j)[u, v]\mathbb{X}_l(j)[u + w, v + h].\end{aligned} \tag{11}$$

The number of parameters of $\mathbb{F}_l$ is $W_{\mathbb{F}}H_{\mathbb{F}}N^2$ and the computation of $\mathbb{F}_l * \mathbb{X}_l$ is of $\mathcal{O}(W_{\mathbb{F}}H_{\mathbb{F}}W_{\mathbb{X}}H_{\mathbb{X}}N^2)$ and $\mathcal{O}(W_{\mathbb{X}}H_{\mathbb{X}}N^2)$ for a standard and a pointwise ($W_{\mathbb{F}} = H_{\mathbb{F}} = 1$) CONV2D, respectively. If a CONV layer is trained on a CSC layer, then $\mathbb{Y}_l(i) = \sum_{j=0}^{F-1} \mathbb{F}_l(i, (i + jD) \bmod N) * \mathbb{X}_l(j)$, that performs zero skipping by computing over only nonzero channels in $\mathbb{F}_l$ that are dilated by D_l channels. Thus, the number of parameters is $W_{\mathbb{F}}H_{\mathbb{F}}NF$ and the computation is reduced to $\mathcal{O}(W_{\mathbb{F}}H_{\mathbb{F}}W_{\mathbb{X}}H_{\mathbb{X}}NF)$ for one single $\mathbb{F}_l * \mathbb{X}$. Given L homogeneous CSC layers with size N, fan-out F, and connectivity C that on every synapse of which 1×1 kernels are laid, the computation of $\mathbb{F}_T * \mathbb{X} = \mathbb{F}_0 * \mathbb{F}_1 * \ldots \mathbb{F}_{L-1} * \mathbb{X}$ is of $\mathcal{O}(W_{\mathbb{X}}H_{\mathbb{X}}NF\log_F(NC))$. Finally, $\mathbb{F}_l$ can be compressed

```
for i = 0 : N − 1
  for u = 0 : W_Y − 1
    for v = 0 : H_Y − 1
      for j = 0 : F − 1
        for w = 0 : W_F − 1
          for h = 0 : H_F − 1
            $\mathbb{X}_{temp} = \mathbb{X}[(i + D\times j) \bmod N,\ u + w, v + h]$
            $\mathbb{Y}[i, u, v] \mathrel{+}= \mathbb{F}[i, j, u, v] \times \mathbb{X}_{temp}$
```

Fig. 5. Naive pseudocode snippet to implement one CSC-CONV layer using a channel-first format for the three tensors $\mathbb{X}$, $\mathbb{F}$, and $\mathbb{Y}$.

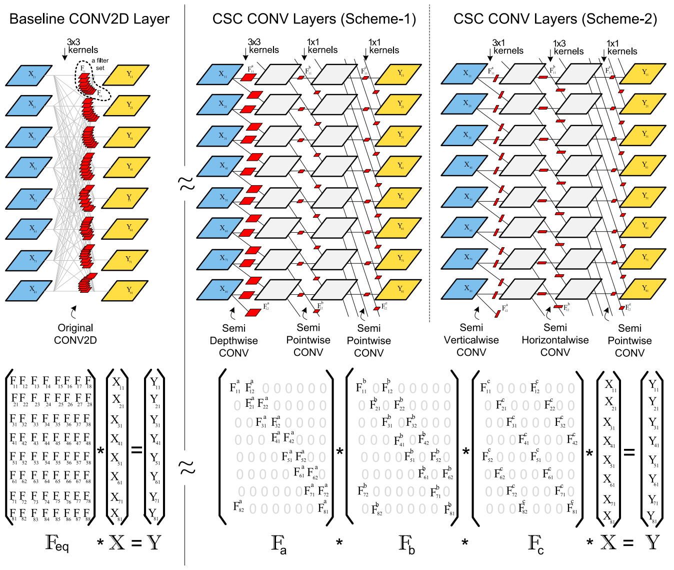


Fig. 6. Two schemes for CONV layers: in Scheme-1, $K \times K$ kernels are laid on the first layer of the CSC architecture and 1×1 kernels for the rest of the layers. In Scheme-2, $K \times 1$ and $1 \times K$ kernels are laid on the first two layers of the CSC architectures and 1×1 kernels for the rest.

format $\hat{\mathbb{F}}_l$, where $\hat{\mathbb{F}}_l(i, j) = \mathbb{F}_l(i, (i + jD_l) \bmod N)$, and the computation of the output $\mathbb{Y}_l$ can be reformulated as $\mathbb{Y}_l(i) = \sum_{j=0}^{F-1} \hat{\mathbb{F}}_l(i, j)*\mathbb{X}_l((i + jD_l) \bmod N)$, plugging which in (11) results in a CSC CONV layer operation, with argument D_l as dilation, between a compressed 4-D filter tensor $\hat{\mathbb{F}}$ of shape $W_F \times H_F \times N \times F$ and a 3-D input tensor $\mathbb{X}$ of shape $W_X \times H_X \times N$ as follows:

$$\begin{aligned}\mathbb{Y}_l(i)[u, v] &= (\mathbb{F}_l * \mathbb{X}_l)(i)[u, v]\\ &= \sum_{j=0}^{F-1}\sum_{w=0}^{W_F-1}\sum_{h=0}^{H_F-1} \hat{\mathbb{F}}_l(i, j)[u, v]\\ &\quad \times \mathbb{X}_l((i + jD_l) \bmod N)[u + w, v + h]\end{aligned} \tag{12}$$

to which, similar to (10), we refer as a cyclic channel dilated 2-D convolution for a CSC-CONV layer. A pseudocode to implement (12) is shown in Fig. 5. For $F = N$ and $D = 1$, (12) converts to a standard 2-D convolution as in (11). For $F = 1$ and $D = 0$, they convert into a depthwise convolution, highlighting the application span of convolution layers from the perspective of cyclic channel dilated 2-D convolution.

We define two schemes for CONV layers factorized on CSC architecture, as shown in Fig. 6. In Scheme-1, the $K \times K$ kernels are assigned to the first layer of the CSC architecture and 1×1 kernels for the rest of the layers. In Scheme-2, $K \times 1$ and $1 \times K$ kernels are designated to the first and the second layers, respectively, and 1×1 kernels for the rest of the layers of the CSC architectures.

C. Bottom-Up Training

The DCNN model with FC/CONV layers is trained first and a reference accuracy $\lambda_{\text{FC/CONV}}$ is obtained. Then, the FC/CONV is replaced with an adjusted CSC-FC/CONV (refer to Section IV-A2), starting from the most compressed CSC architecture and directed toward the compression reduction. Technically, (3) indicates that the highest compression for homogeneous CSC architectures is achieved given $C = 1$ and $F = e$, where e is the Napier's constant ($\approx$2.718). With no strict definition, however, we consider that the most compressed CSC architecture is given by small values for F (e.g., 2, 3, and 4) if adopting a CSCI architecture and clearly by $C = 1$ if adopting a CSCII architecture. The experiments begin by targeting the bulky layers of a DCNN. In each experiment, if the accuracy λ_{CSC} is ϵ less than $\lambda_{\text{FC/CONV}}$, the procedure is terminated, and the CSC model is accepted. If no CSC layer replacement satisfies the accuracy loss, the original layer is restored. The criterion ϵ is chosen to be 2% in all experiments of this article. While we do not particularly promote either of CSCI or CSCII over another, we observe that CSCI architectures provide the most compression for a shallow network such as LeNet, and CSCII architectures provide the most accuracy stability for deeper networks such as AlexNet and MobileNet.

D. Similar Works

1) PermDNN: Yang *et al.* [21] proposed PermDNN that is an approach to customizing sparse weight matrices with a diagonalization scheme. In one glance, the CSC layers in this work look similar to the permuted diagonal matrices in [21]. However, in PermDNN, a weight matrix is chunked into many smaller diagonalized matrices and the connectivity of consecutive layers is not defined as a tweakable parameter as in CSC layers. On the other hand, a CSC layer uses one continuous diagonalized (cyclic) weight matrix per layer, the cascade of a few of which guarantees the full connectivity between alternate layers.

2) CIRCNN: In CIRCNN [32], a weight matrix is partitioned into $K \times K$ submatrices each of which, due to their circulant structure, can be defined with only K weights, thus compressing the model by K times. The convolution operation of the block-circulant matrices can be efficiently performed using FFT, elementwise matrix multiplication, and inverse FFT operations. With all the complexity reduction advantages of the CIRCNN, it has two drawbacks: 1) the computation needs to be carried out in the realm of complex numbers and 2) the method cannot be used in tandem with extreme quantization, as the precision of the quantized weight matrices does not propagate in their transformation into Fourier domain.

VI. Experiments and Results

We used CSC architectures to evaluate their compression impact for popular DCNNs, including LeNet-300-100 [5], AlexNet [33], and MobileNet [3] on benchmarking datasets MNIST and ImageNet. To follow the bottom-up training procedure, for each experiment, one of the two CSC architectures and one of the two proposed CSC-CONV schemes from Section V is adopted and only one CSC hyperparameter is tweaked per experiment. We use the AlexNet to show the generalization of our method for compressing all of its CONV and FC layers and for comparison against similar compressing methods. However, for LeNet and MobileNet, we only tackle those layers whose compression has an overall impact on either the size or the computation of the model. These layers include the first two and the last two layers of LeNet300100 and MobileNet, respectively.

TABLE I

Baseline AlexNet and Its Compressed Model Using CSCII-Scheme-1 in Detail

AlexNet	Original			**Compressed by CSC Scheme-1**		
Layer	Input Shape $W_{\mathbb{X}}\times H_{\mathbb{X}}\times N_I$	Filter Shape $W_{\mathbb{F}}\times H_{\mathbb{F}}\times N_I\times F$	Dial. D	Input Shape $W_{\mathbb{X}}\times H_{\mathbb{X}}\times N_I$	Filter Shape $W_{\mathbb{F}}\times H_{\mathbb{F}}\times N_I\times F$	Dial. D
Conv1_1/S4	227×227×3	11×11×3×96	1	227×227×3	11×11×3×16	1
Conv1_2	-	-	-	55×55×96	1×1×96×96	1
Conv2_1	27×27×96	5×5×96×128	2	27×27×96	5×5×96×32	1
Conv2_2	-	-	-	27×27×256	1×1×256×128	2
Conv3_1	13×13×256	3×3×256×384	1	13×13×256	3×3×256×64	3
Conv3_2	-	-	-	13×13×384	1×1×384×192	2
Conv4_1	13×13×384	3×3×384×192	2	13×13×384	3×3×384×24	1
Conv4_2	-	-	-	13×13×384	1×1×384×192	2
Conv5_1	13×13×384	3×3×384×128	2	13×13×384	3×3×384×24	1
Conv5_2	-	-	-	13×13×384	1×1×384×128	2
FC6_1	6×6×256	6×6×256×4096	1	6×6×256	6×6×256×256	1
FC6_2	-	-	-	1×1×4096	1×1×4096×512	8
FC7_1	1×1×4096	1×1×4096×4096	1	1×1×4096	1×1×4096×256	1
FC7_2	-	-	-	1×1×4096	1×1×4096×256	16
FC8_1	1×1×4096	1×1×4096×1000	1	1×1×4096	1×1×4096×160	1
FC8_2	-	-	-	1×1×4000	1×1×4000×100	10
Params (imp.)	60,954,656 (ref.)			8,243,504 (7.4×)		
Ops (imp.)	1,448,813,632 (ref.)			438,227,040 (3.3×)		
Top-1 Acc.	57.1%			55.9%		
Top-5 Acc.	80.2%			79.1%		

TABLE II

Comparison Between Compressing Methods to Ours for AlexNet

Layer	Param.	Pruning [15]	Pruning [34]	Pruning [35]	Structured [27]	Compact [21]	**Compact (ours)**
CONV1	35K	84%	54%	83%	100%	100%	43%
CONV2	307K	38%	41%	26%	79%	100%	36%
CONV3	885K	35%	28%	23%	60%	100%	25%
CONV4	664K	37%	32%	23%	60%	100%	23%
CONV5	443K	37%	33%	23%	75%	100%	30%
FC6	38M	9%	4%	7%	100%	10%	12%
FC7	17M	9%	7%	7%	100%	10%	13%
FC8	4M	25%	5%	18%	100%	25%	26%
Total	61M	11%	6%	8%	99%	14%	14%
FLOPs	1.5B	30%	25%	24%	76%	93%	30%
Top-1 Acc.	57.1%	-	-	-	57.5%	-	55.9%
Top-5 Acc.	80.2%	80.2%	80.0%	80.4%	-	80.0%	79.1%

3) AlexNet: For AlexNet, all CONV and FC layers were replaced with CSCII architectures using Scheme-1 for CONV layers, as shown in Fig. 6. The baseline architecture and the compressed model are elaborated in detail in Table I with max-pool layers omitted. Table I also translates the original AlexNet in terms of CSC layers where the second, third, and fourth CONV layers are dilated for dispatching to two separate GPUs. The number of parameters in every layer of the two models is $W_{\mathbb{F}} H_{\mathbb{F}} N_I F_{\mathbb{F}}$ and the number of operations is 2 $W_{\mathbb{X}} H_{\mathbb{X}} W_{\mathbb{F}} H_{\mathbb{F}} N_I F_{\mathbb{F}}/S^2$ and $2(W_{\mathbb{X}} - W_{\mathbb{F}} + S)(H_{\mathbb{X}} - H_{\mathbb{F}} + S) W_{\mathbb{F}} H_{\mathbb{F}} N_I F_{\mathbb{F}}/S^2$ for layers with same and valid size operations, respectively. The stride S is 4 for the layer Conv1_1 and is 1 for others. The layers Conv1_1 and FC6_1 are of valid size and the rest are of the same size operations. For the compressed AlexNet, the connectivity of most of the CSCII layers is 16, and the model is exactly equivalent to an original AlexNet with different weights, but with factorizable layers each conforming to two sparse layers in accordance to CSCII-Scheme-1. Table II compares our compressed AlexNet with related compressing methods, indicating that the structurally sparse CSC architectures can compress all CONV and FC layers of the AlexNet on par with pruning methods.

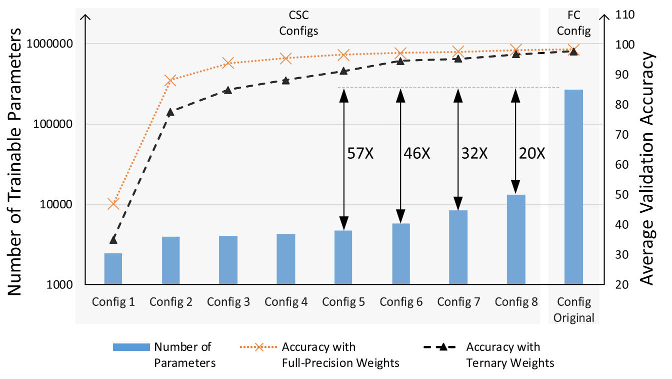


Fig. 7. Impact of the bottom-top method of replacing FC with CSC layers on the accuracy of LeNet-300-100 trained with full and ternary-precision weights. In each configuration, FC1 and FC2 are replaced with two CSCI graphs, CSCI1 and CSCI2, and the last layer remains FC. For configuration 1, CSC layers are one-node factorized layers. For configuration i ($i > 1$), CSCI1 has parameters $C_1 = 1$, $F_1 = 2$, $L_1 = i + 1$, and $N_1 = 2^{i+1}$, and CSCI2 has parameters $C_2 = 1$, $F_2 = 2$, $L_2 = i$, and $N_2 = 2^i$. In total, each configuration ($i > 1$) has $E_T = 2^i(6i - 8) + 3968$ parameters.

4) LeNet-300-100: For LeNet-300-100, which is a three-layer DNN, we replaced the first two FC layers that are the most memory- and compute-intensive layers, with CSCI architectures. Fig. 2(a) and (c) shows the baseline model and the compressed model of the LeNet-300-100 DNN. Because $F^L = NC$ should hold for all homogeneous CSC architectures according to (2) and since adopting a CSCI architecture already narrows one of the four hyperparameters, i.e., $C = 1$, here, the bottom-up training method means increasing any of F, L, or N such that $F^L = N$ holds. For simplicity, in our exploration, we took $F = 2$ (which is close to Napier's constant) and increased L (thus $N = 2^L$) in every incremental experiment until the accuracy loss of less than 2% was met. Table III summarizes the compressed 32-bit and model and compares it with the baseline and a related pruned model.

In another set of experiments, we altered the MNIST dataset to a thresholded MNIST dataset in which the 28×28 grayscale pixels from the original MNIST were converted to ternary values, i.e., -1, 0, and $+1$. Then, we further compressed the LeNet-300-100 by ternarizing its weights and fmap using the QKeras libraries [25], once for the baseline DNN enforcing 95% zero values, and another time for the CSC compressed model. For the former compression, we increased the thresholding parameter in ternary layers of the ternary DNN to the extent to have 95% zero values. We refer to the latter compression method as a double-compression method since both CSC architecture and the extreme quantization independently reduce the model size of the DNN by 16× and 19×, respectively. Meanwhile, the computation is lowered by 19× and simplified to ternary operations. The ternarized baseline model can either be stored with all its parameters and with no compression scheme in which 2 bits are dedicated to each of the 267k parameters or it can be represented with a compressing format such as COO (coordinate) format that maintains only the 5% nonzero weights, 1 bit dedicated to each, as well as their indices, and 19 bits for each index. Table IV summarizes the compressed 2-bit models. Fig. 7 shows the bottom-up training procedure by tweaking the parameter L for each CSCI architecture for the first two layers of the DNN. In Section VII, we will implement the ternarized DNNs to further shed light on the advantages gained by ternarization and structured sparsity.

TABLE III

LeNet-300-100 With 32-Bit Floating-Point Model as the Baseline, Its Ternarized, and Their CSCI Compressed Models, Compared to a Related Work. * Indexing Overhead Is Not Accounted for Size of the Fine-Grained Model

* The indexing overhead is not accounted for size of the fine-grained model.

LeNet-300-100 (MNIST)	Layer Sparsity 235K-30K-1K	Total Params	#Ops (K)	Acc. %	Model Size (Compression)
Baseline	100%-100%-100%	267K	534	98.4	1065 KB (1×)
Compressed w/ CSCI	**4%-13%-100%**	**14K**	**28**	**98.2**	**53 KB (19×)**
Fine-grained Pruning [15]	8%-9%-26%	22K	44	98.4	88* KB (12×)

TABLE IV

Ternarized LeNet-300-100 With 95% Zero Values Represented With Either No Compression or COO Format, as Well as CSCI Compressed LeNet-300-100. * Indexing Overhead Is Accounted for Size of the COO Model

* The indexing overhead is accounted for size of the COO model.

LeNet-300-100 (Thresholded MNIST)	Layer Sparsity 235K-30K-1K	Total Params	#Ops (K)	Acc. %	Model Size (Compression)
Ternarized	100%-100%-100%	267K	534	97.1	66.7 KB (1×)
Tern. & pruned (COO)	5%-5%-50%	14K	28	97.1	33.3* KB (2×)
Ternarized & CSCI	**4%-13%-100%**	**14K**	**28**	**96.7**	**3.5 KB (19×)**

TABLE V

0.5 MobileNetV1-192 With 32-Bit Floating-Point Model as the Baseline, Its 8-bit Quantized, and Compression Over Their Last Two Layers Using CSCII

0.5 MobileNetV1-192 (ImageNet)	Model Precision	Total Params	#Ops (M)	Top-1 Acc.(%)	Top-5 Acc.(%)	Model Size (Compression)
Baseline	32-bit	1320K	220	61.7	83.6	5.3 MB (1.0×)
CSCII for last 2 layers	**32-bit**	**873K**	**210**	**60.3**	**82.4**	**3.5 MB (1.5×)**
TFlite Model	8-bit	1320K	220	60.0	82.2	1.4 MB (4.0×)
CSCII for last 2 layers	**8-bit**	**873K**	**210**	**58.7**	**81.1**	**0.9 MB (6.0×)**

5) MobileNet: MobileNets are compact models already, and the depthwise/pointwise layers are special cases of heterogeneous CSC architectures; all depthwise layers in MobileNets can be processed given $F = 1$ and $D = 0$ and all pointwise layers can be processed given $F = N$ and $D = 1$ with reference to (12). Nevertheless, in all of MobileNets' variations, more than 50% of the model size falls under the last two layers, i.e., a pointwise CONV and an FC layer. We initially attempted to compress all layers of the MobileNets to ultimately find out that only its two last bulky layers have the potential to be further compressed with little accuracy loss using our method. In this experiment, we replaced the last two layers of MobileNets-224 and -192 with CSCII architectures and determined $C = 16$ for each CSCII architecture. To facilitate our experiments, we used pretrained 8-bit TFlite models and used transfer learning to fine-tune the last two CSCII layers. Fig. 8 plots the number of parameters versus the ImageNet top-1 accuracy in the 8-bit quantized models of 0.5, 0.75, and 1.0 MobileNetV1-224 (blue circle markers) and MobileNetV1-192 (red circle markers), and their CSCII compressed counterparts (triangle markers). The compressed CSC-MobileNet-224 and CSC-MobileNet-192 have above AlexNet-level accuracy and are small enough (873 kB) to be storable on a memory of less than 1 MB. Table V summarizes the experiment for 0.5 MobileNet: the compressed quantized model of 0.5 MobileNet is 6× and 1.5× smaller than its float-32 and 8-bit baselines. In Section VII, we will implement the compressed 8-bit 0.5 CSC-MobileNet-192 on a tiny Xilinx Artix-7 FPGA to elucidate the benefits gained by full on-chip processing of a further-compressed DCNN.

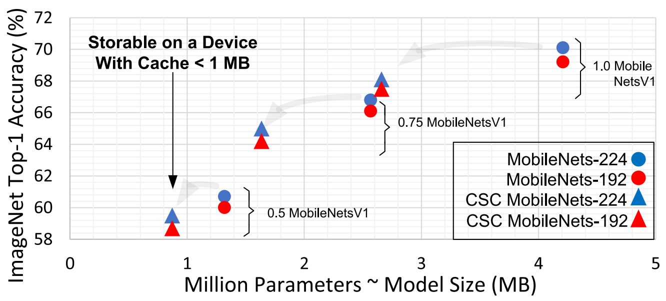


Fig. 8. Number of parameters (∼model size) versus top-1 ImageNet accuracy of 8-bit quantized DCNNs including 0.5, 0.75, and 1.0 MobileNetV1-224 (blue circle markers), and 0.5, 0.75, and 1.0 MobileNetV1-192 (red circle markers) and their CSC compressed counterparts (triangle markers). The CSC compressed counterparts are resulted by compressing the last two layers of every original MobileNet using CSCII architectures.

VII. Accelerator Architecture and Implementation

Motivated by works of Zhang *et al.* [1] and Han *et al.* [36], [37] that emphasize the importance of compressing DCNNs to implement near SRAM processing, we pursue designing hardware for FPGAs that is independent of a DRAM and that integrates all the processing components along with sufficient on-chip SRAM memory that stores the whole DCNN model and its intermediate feature map during every inference. The peak performance in a well-engineered accelerator with #PE processing engines and $\#\text{MAC}_{\text{per PE}}$ MAC units per PE with a clock rate freq can potentially approach $2 \times \#\text{PE} \times \#\text{MAC}_{\text{per PE}} \times \text{freq}$. According to the roofline model, this performance is achievable if the memory bandwidth is high enough to minimize stalls and to continuously feed the computing components in PEs with required data/operands. All our hardware configurations in this section were implemented on Xilinx Artix-7 FPGAs and all power and latency analyses were measured using the Xilinx Vivado Design Suite.

A. Ternarized LeNet-300-100 for Thresholded MNIST

To further illuminate the importance of structured sparsity for extremely quantized networks, in this section, we first evaluate the ternarized baseline LeNet-300-100 on two hardware designs: a hardware that implements the ternarized DNN with no compression scheme and a hardware that implements the DNN using coordinate (COO) format. We then compare the two implementations with a third hardware that implements the DNN compressed with CSC architectures. The three designs are implemented with one PE and have one input memory that is large enough to store one 28×28 thresholded MNIST image of size 196 B, an output memory to accommodate the largest fmap of size 75 B, and one (or two) memory units to store the DNN models adapted to the three ternarized LeNet DNNs as in Table IV. Fig. 9(a) shows the block diagram of the CSC matrix–vector multiplication processor, which implements (8) with ternary values for the baseline ternarized LeNet, where the three weight matrices from the three layers of the DNN of size 66.7 kB are stored using row-major order in 32 36-kB BRAMs constituting the weight memory. Fig. 9(b) implements the same DNN but using a COO format for only the 5% nonzero values within the ternarized model. This design incorporates a weight memory that holds 1-bit data values (+1 and −1) and an index memory

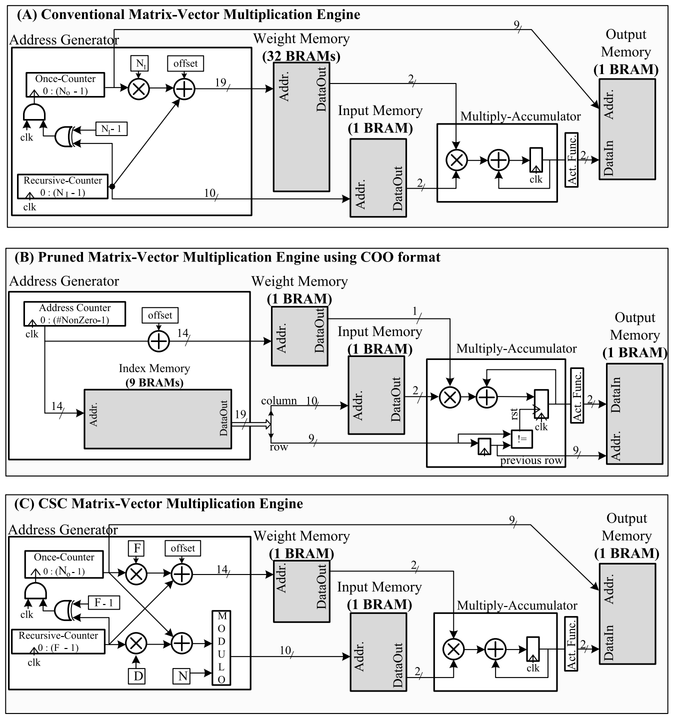


Fig. 9. Deploying the ternarized LeNet-300-100 DNNs on (a) hardware that implements the baseline model, (b) hardware that implements the baseline model, but compressed with COO format, and (c) hardware that implements the DNN compressed with CSC architectures.

TABLE VI
LeNet-300-100 With 2-bit Models Implemented on Three Hardware: Baseline, Pruned Compressed With COO, and Compressed With CSC

LeNet-300-100 (Threshold. MNIST)	Layer Sparsity 235K-30K-1K	LUT (#)	BRAM (#)	Delay (uS)	Dyn. Pow. (mW)	Total Pow. (mW)	Dyn. En. (uJ)	Total En. (uJ)	MNIST Acc. (%)
Ternarized	100%-100%-100%	698	34	2690	31	90	83	242	97.1
Tern. & Pruned	5%-5%-50%	910	12	140	66	125	9	18	97.1
Ternarized & CSC	4%-13%-100%	962	**3**	140	**17**	76	**2**	**11**	96.7
Device (XC7A35T)		14,600	45						

that store a 19-bit index per nonzero value, 10 and 9 bits of which are column and row pointers of the nonzero weights in the pruned layer, respectively, thus requiring an extra ∼14k × 19-bit index memory that is instantiated by 9 36-kb BRAMs. The column pointers address the input memory, whereas the row pointers address the output memory.

Fig. 9(c) shows the block diagram of the CSC matrix–vector multiplication processor, which implements (10) for weight matrices of the CSC-FC layers of the DNN that are stored in memory in accordance to their compressed format $\hat{\mathbf{W}}$ in a row-major order. The schematic is adapted to accommodate the LeNet300100 with CSCI layers that have a model size ∼14 kB (Config 7) and that requires only one BRAM to be instantiated. The address generator unit generates the indexes for the weight matrix and the input data using two counters with reference to (10) and is updated with the parameters and the starting address (layer offset) of the layer-under-process. The address counter counts from 0 to $N_I - 1$ only once, while the recursive counter counts from 0 to $F - 1$ recursively until the layer is processed thoroughly. A state machine updates the address generator with the layer's parameters, controls the conditions of tiled/truncated layers, and swaps the input and output memory units alternately.

Implemented on one of the smallest Xilinx Artix-7 FPGAs, the performance of all three designs with only one MAC unit at a clock rate of 100 MHz is 200 MOPS. Thus, with reference to Table IV, having the highest computation, the first design takes has the longest classification latency per inference. The second and the third design have relatively the same number of operations, thus having an equal execution time of 140 μs. However, due to having more memory usage and more energy-consuming communication, the pruned DNN implementation has approximately $4\times$ more dynamic power and dynamic energy compared to those of the CSC compressed DNN implementation. Table VI summarizes the three implementations on a Xilinx XC7A35T FPGA.

B. CSC Compressed MobileNet for ImageNet

In Section V, we showed that DCNN's major layers, including standard FC, standard CONV, depthwise separable CONV, as well as CSC-FC and CSC-CONV layers, are all special cases of CSC layers and can be recognized solely by their weights in compressed formats ($\hat{\mathbf{W}}$ or $\hat{\mathbb{F}}$) and hyperparameters N (and N_I and N_O if adjusted), F, D, $W_{\mathbb{F}}$, and $H_{\mathbb{F}}$, and their computation are all governed by (12). The computation in (12) with a pseudocode shown in Fig. 5 for a CSC-CONV can be highly optimized using various computation tiling schemes for any parallel computing machine. Our proposed hardware for the compressed MobileNet is shown in Fig. 10 and described in Verilog HDL. It computes DCNNs with or without CSC layers and is configurable with a number of PEs, a number of MAC units per PE, and the dataflow bitwidth matching the quantization level of a DCNN model. The hardware comprises three main blocks, including an array of PEs that perform MAC operations and accommodate a DCNN weight model partitioned in weight memory units, a partitioned input memory that stores intermediate input feature map (ifmap) data, and a CSC-based router that links and maximizes the bandwidth between array of PEs and array of subbanks in the ifmap memory. Each PE also incorporates another SRAM memory that stores output feature map (ofmap) data in partitions that exchange its temporary data with the ifmap memory to drive a next layer in the DCNN. For simplicity, the state machine and pipelines that perform batch normalization, max pool, quantization, and nonlinear activations are not shown. We configure this hardware to be implemented on a resource-bound tiny FPGA and to deploy the 0.5 CSC-MobileNet-192 for full on-chip processing. The computation tiling scheme of the hardware to implement DCNN layers governed by (12) will be further elaborated in the following.

1) Quantization: We follow the quantization scheme of the TensorFlow lite set of tools to effectively reflect their underlying computation on to the fixed-point pipelines of our hardware. In a nutshell, in 8-bit quantized TFlite models, both the weight tensor $\mathbb{F}$ and the input fmap $\mathbb{X}$ are in 8-bit precision and the convolution operation between the two operand tensors followed by a quantizing rectified linear unit (ReLU) should result in another tensor $\mathbb{Y}$ that has 8-bit values. In order to do so, for a given layer with 8-bit quantized weights $\mathbb{F}$ and all possible 8-bit quantized values of $\mathbb{X}$ that are resulted from the dataset, an 8-bit offset scalar α and a 32-bit scaling factor scalar β are calculated and fine-tuned based on the min/max values of $\mathbb{Y}$ during the training. Taking both scalars into account, the operation $\mathbb{F} * \mathbb{X}$ as in (9)–(12) is converted into $\beta(\mathbb{F} - \alpha) * \mathbb{X}$, the most significant 8 bits of rounded entries of which determine the quantized $\mathbb{Y}$.

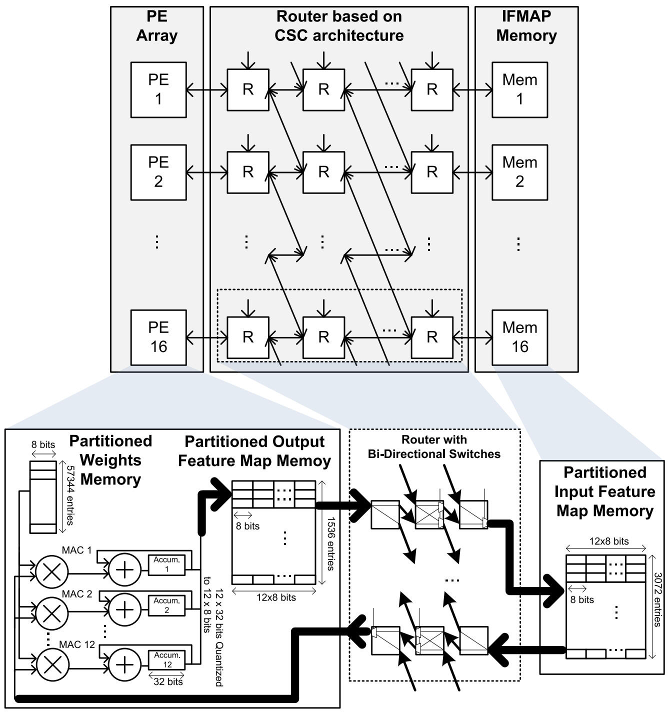

PE Array
Router based on CSC architecture
IFMAP Memory
PE 1
PE 2
PE 16
R
Mem 1
Mem 2
Mem 16
Partitioned Weights Memory
8 bits
57344 entries
Partitioned Output Feature Map Memoy
8 bits
1536 entries
12x8 bits
MAC 1
MAC 2
MAC 12
Accum. 1
Accum. 2
Accum. 12
32 bits
12 x 32 bits Quantized to 12 x 8 bits
Router with Bi-Directional Switches
Partitioned Input Feature Map Memory
12x8 bits
8 bits
3072 entries

Fig. 10. Accelerator hardware design to implement standard, depthwise-separable, and CSC-compressed DCNNs, configurable with a number of PEs, a number of MAC units per PE, and dataflow bitwidth, highlighting array of PEs and IFMAP memory units interlinked with a high-bandwidth router.

2) Tiling Scheme: For the computation of convolution operations as in (11) and (12), there exists at least three different parallel computation schemes, namely input channel tiling, output channel tiling, and image patch tiling. Fig. 11 shows the three schemes illustrating four PEs that parallelly compute the standard 2-D convolution between a four-channel ifmap and four sets of filters. In our work, the fmap data are stored using an image-to-column format where each column from the 2-D channel of an fmap is folded and stored in one or multiple entries of a partitioned fmap memory. The kernels are stored in the partitioned weight memory using a row-major order for both 3×3 and 1×1 kernels. The array of PEs adopts an output channel tiling scheme in accordance to Fig. 11(a), i.e., each PE concurrently performs a 2-D convolution between one channel from the input fmap that is accessible through the CSC-based router and one individual filter channel that is stored in the partitioned weight memory of the PE. Meanwhile, each PE incorporates multiple MAC units that adopt an image patch tiling scheme, in accordance to Fig. 11(c), to parallel compute a single 2-D convolution by partitioning a fetched 2-D channel along its rows into a number of MAC units to tile the computation. Upon fetching each entry from a partitioned fmap memory, one or a fraction of an fmap column goes through the pipeline of the MAC units to be individually multiplied with one common weight value fetched from the partitioned weight memory and accumulated in different accumulators. Tiling the input image in that regards has no overhead for 1×1 kernels but has little timing overhead for 3×3 kernels that constitute less than 5% of the total computation in the compressed MobileNet. To handle the 3×3 kernels for an image patch tiling scheme, the tiled partitions of the fmap channels have overlapping data from the marginal rows at which the fmap channel is tiled. The state machine

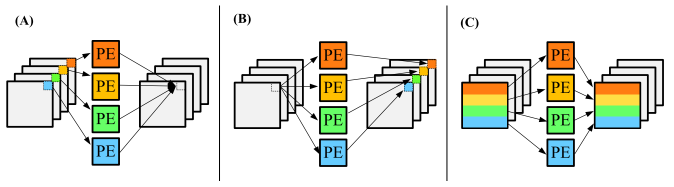


Fig. 11. Computation tiling schemes for 2-D convolution: (a) input channel tiling, (b) output channel tiling, and (c) image patch tiling.

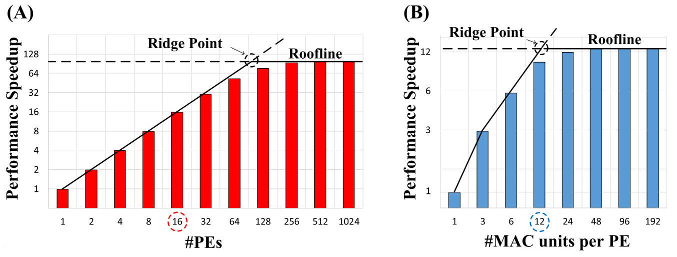


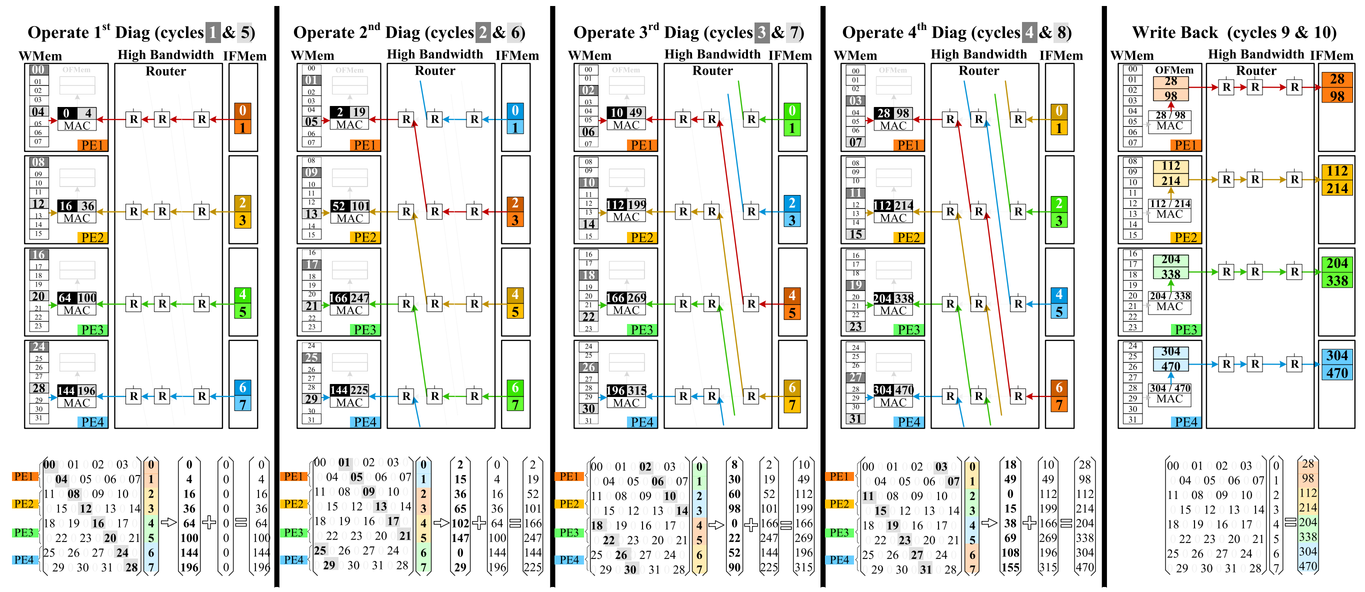
Fig. 12. Roofline model for our hardware to implement 0.5 MobileNet-192 as a workload under different hardware configurations. (a) Speed-up versus #PE adopting an output channel tiling scheme. (b) Speed-up versus #MAC$_{\text{per PE}}$ adopting an image tiling scheme.

synchronizes the overlapping marginal rows for the output fmap of only the layers that are followed by their next layer with 3×3 kernels.

3) Roofline Model: The roofline model is used to capture the speedup limitations of hardware and varies for different workloads and implementation styles. We use a naive model based on our workload 0.5 CSC-MobileNet-192 and our implementation style, which is multiple MAC units per PE adopting image channel tiling and array of PEs adopting output channel tiling. We tweak the parameters #PE and #MAC$_{\text{per PE}}$ to explore the maximum capacitance of our design and to pick a configuration that assures no hardware resource redundancy. Fig. 12(a) shows that by increasing the number of PEs, the performance speeds up linearly until a number of PEs approach an amount of 64, which indicates that in the workload 0.5 CSC-MobileNet, the majority of the computation is over layers that have more than 64 channels. Similarly, Fig. 12(b) shows that by increasing #MAC$_{\text{per PE}}$ until meeting an amount of 12, the performance speeds up almost linearly, which is because, in the workload 0.5 MobileNet-192, the majority of the computation is over layers that have fmap channels of size more than 12×12 that can perfectly be tiled into 12 patches along their rows and can be handled using 12 MAC units. Having assumed that the hardware memory is unlimited in both Fig. 12(a) and (b), the maximum speedup approaches to 96× and 13×, respectively. Since we are seeking an energy-efficient design, rather a maximum achievable performance, we select 16 PEs and 12 MACs per PE because they are both on the linear region of their roofline models that guarantee avoidance of underloading the computation pipeline, and as close to the ridge point as possible, where the performance is maximum using minimum hardware resource utilization. Another reason to prefer 16 PE over 32 or 64 PEs is that each partitioned input memory should be large enough to store one of the three channels of the input RGB image that is of size 37 kB (=192×192 B) for the first layer and should also be able to accommodate the partitioned largest fmap of the MobileNet-192 that is of size 590 kB (=$96 \times 96 \times 64$ B). Taking 590 kB for the total size of the input fmap memory and partitioning it into 16×3072 B in accordance to 16 PEs meets the implementation requisites for deployment of the compressed 0.5 CSC-MobileNet-192. Consequently, given the choice of 12 MAC units per PE, each partitioned input memory has a width 12×8 bits and a depth 3072 that is instantiated using 8.5×36 kB BRAMs.

4) High-Bandwidth Router Adopting Cyclic Architectures: The DCNN's compressed parameters $\hat{\mathbb{F}}$ given a layer and its input feature map $\mathbb{X}$ are partitioned and evenly distributed between #PE weight memory subbanks and #PE input fmap memory subbanks, respectively, i.e., given a layer with N nodes/channels fed by an N-channel input fmap, the $W_{\hat{\mathbb{F}}} H_{\hat{\mathbb{F}}} N F$ parameters of $\hat{\mathbb{F}}$ and the $W_{\mathbb{X}} H_{\mathbb{X}} N$ of fmap are equally partitioned and stored in #PE weight memory subbanks and #PE input fmap subbanks, respectively, on a channel-major order. Each of the PEs is designated to compute (1/#PE) of the total computation attributed to the layer and should have direct access to one and only one partitioned weight memory at all times and can be provided with access to each and every #PE memory subbanks at different computation cycles. To effectively tile the computation for the layer and link the individual input fmap memory subbanks with the PEs, we designed a router that adopts a cyclic architecture that essentially implements the modulo operator existing in (12). The cyclic router is composed of switches that are controlled by a state machine driven with the layer's hyperparameters to provide a one-to-one cyclic link between ifmap subbanks and MAC units from the PEs. Fig. 13 reflects our idea of tiled computation by illustrating operation between an 8×8 CSC-FC matrix ($N = 8, F = 4$, and $D = 2$) and a vector of size 8, using a hardware configured with four two-word input subbanks, four PEs (with 1 MAC unit), four eight-word weight, and four two-word output subbanks. The total computation for these examples is equal to 32 MAC operations, which is carried out in eight clock cycles using a router with switches that take advantage of the cyclic structure and instantly link the computing resources to required data. Using a total four MAC units to perform 32 MAC operations in eight clock cycles indicates that the design meets the maximum achievable performance. Finally, two more clock cycles are spent to write the eight-word output fmap back to the four input subbanks. To generalize the illustrating example of Fig. 13, consider that each of the values stored in the input fmap subbanks is a 2-D channel from an input fmap. Thus, with a scheme similar to the figure and with reference to (12), all types of DCNN layers, including FC, CONV, depthwise, CSC-FC, and CSC-CONV, are indistinguishably implementable within the same pipeline, all with performance $2 \times \#\text{PE} \times \#\text{MAC}_{\text{per PE}} \times \text{freq}$. For our design with 16 PEs and 16 ifmap subbanks, we adopted a CSC structure for the router with parameters $L = 2$ stages, $N = 16$ switches per stage, and $F = 4$ degree of switches, which provides an internal bandwidth of 19.2 GB/s and performance of 38.4 GOPS at a clock rate of 100 MHz.

5) Hardware Implementation: The hardware configuration for the compressed model was implemented on the Artix 7A200T FPGA from the AC701 evaluation board at a clock rate of 100 MHz using the Xilinx Vivado Design Suite. The measurements of power, latency, and consequently the energy per inference of the workloads come from simulation using corresponding test benches that provide precise toggle rate as well as timing of the implementation on the FPGA. The 7A200T FPGA has 1.6 MB of BRAMs and 0.4 MB of distributed LUT RAMs that suffice the total on-chip SRAM requested for implementing the hardware configuration



Operate 1st Diag (cycles 1 & 5)
Operate 2nd Diag (cycles 2 & 6)
Operate 3rd Diag (cycles 3 & 7)
Operate 4th Diag (cycles 4 & 8)
Write Back (cycles 9 & 10)
WMem
High Bandwidth Router
IFMem
OFMem
MAC
PE1
PE2
PE3
PE4
R

Fig. 13. High-bandwidth router that adopts a CSC architecture to implement a CSC DCNN layer, illustrated with a simple example: multiplication between a cyclic weight matrix with $F = 4$, $D = 2$, $N = 8$, a vector with size 8, using four PEs, performed in eight cycles.

that accommodates the compressed MobileNet, whereas the uncompressed MobileNet requiring hardware with 2.4-MB memory cannot fully fit in the on-chip memory of the FPGA.

The hardware for the compressed MobileNet has been configured as per Fig. 10 with 16 PEs and 12 MACs per PE, with the minimum required on-chip SRAM to implement the 8-bit 0.5 CSC-MobileNet-192 for full on-chip processing. The size and total computation of the DCNN model according to Table V are 873 kB and 210 million operations, respectively. Partitioning the model weights of the 0.5 CSC-MobileNet-192 (of size 873 kB) into 16 results in configuring each partitioned weight memory to have width 8 bits and depth 57 344 that would be instantiated using 14 $\times$ 36-kb BRAMs. A fraction of 3.5% of this memory space is used for storing quantization offset, scaling parameters, and DCNN hyperparameters. The size of each input image is 111 kB (=192 $\times$ 192$\times$3), and the largest feature map is 590 kB (=96 $\times$ 96$\times$64) that is storable in 16 partitioned ifmap subbanks each with size 37 kB (8.5 BRAMs). The largest fmap generates an output feature map of 295 kB (=48 $\times$ 48$\times$128). Equivalently, the output fmap memory should be large enough to store the output fmap of size 295 kB (=96 $\times$ 96$\times$32B) from the layer of the 0.5 CSC-MobileNet-192, which is fed by the largest input fmap. As a result, each partitioned output fmap memory has a width 12 $\times$ 8 bits and a depth 1536, which will be instantiated using 4.5 $\times$ 36 kb BRAMs. Summing up all the memory requirements for the implementation, the FPGA should provide approximately 432 BRAMs. This amount is 18% more than the available resource (364 BRAMs) of our selected Xilinx FPGA. The extra memory requirement is then instantiated using the sufficient LUT RAMs of the FPGA that are of size 0.4 MB. Table VII lists the configuration of the hardware that requires approximately 2 MB on-chip memory to implement ImageNet with top-5 classification accuracy of 81.1%. Table VIII (last column) shows the resource utilization break down for the implementation on the 7A200T FPGA.

With 192 MAC units operating at 100 MHz, the peak performance of the accelerator reaches 38.4 GOPS, and the power dissipation of the FPGA is approximately 1.4 W, thereby yielding an energy efficiency of 27 GOP/J. Fig. 14 shows the power dissipation breakdown per FPGA's resources and the resource utilization for the hardware with 16 PEs and 12 MAC units per PE on the evaluation board, highlighting the full utilization of the BRAMs and dynamic power dissipation of 90%. Fig. 15 shows the number of computation and communication cycles per layer implementation of the 0.5 CSC-MobileNet-192 as workload. With reference to Fig. 13, the computation cycles of a DCNN's layer correspond to the phase in which the MAC units in hardware perform the MAC operations of the layer, and the communication cycles of the layer correspond to the write-back phase in which the processed output fmap is relocated to the input fmap memory to start the computation of the next layer of the DCNN.

TABLE VII

HARDWARE CONFIGURATION FOR 0.5 CSC-MOBILENET-192

Hardware Config.	**0.5 CSC-MobileNet-192**
No. of PEs	16
Total No. of MAC Units	16×12
Input fmap SRAM (B)	16×36864
Weight SRAM (B)	16×57344
Output fmap SRAM (B)	16×18432

VIII. COMPARISON

Table VIII summarizes the FPGA implementation results of our compressed MobileNet for the classification of the ImageNet dataset and compares it with related FPGA implementations that are tabulated from left to right of the table in an efficiency (Inference/J) ascending order. While the DCNN models differ in their architectures, they all adopt some sort of a redundancy-reduction technique in tandem with various quantization schemes, and they all seek the same task, i.e., ImageNet classification with a top-1 accuracy ranging between 50% and 70%. Compared to the baseline MobileNet implementation in the work of Liao *et al.* [13], our implementation has 4.6% less accuracy but gains $\sim$ 1.5$\times$, 100$\times$, and 150$\times$ in terms of power, FPS, and efficiency at clock rate 100 MHz, respectively. Also, in comparison to the works of Zhu *et al.* [12] and Lu *et al.* [40] that adopt an AlexNet

TABLE VIII
COMPARISON WITH RELATED FPGA WORKS FOR IMAGENET CLASSIFICATION

Related Work	[13]	[38]	[39]	[40]	[41]	[14]	[42]	[12]	**This work**
Model Name	0.5 MobileNet-224	DiracDeltaNet	ResNet-18	AlexNet	DoReFaNet/PF	MobileNet	1.0 MobileNet-224	AlexNet	0.5 MobileNet-192
Method	Compact (Baseline)	Compact Model	Ternarized	Pruned	Hybrid Quantization	Shallow Network	Redundancy-Reduced	Structurally Sparse	CSC for the Last 2 layers
Precision (W/A)	16/16	4/4	2/2	16/16	1/2	4/4	8/4	16/16	8/8
Top-1 Acc.	63.3	68.3	65.6	57.1	50.3	-	64.6	57.3	58.7
Top-5 Acc.	84.9	88.1	-	80.2	-	-	84.5	-	81.1
Device	7Z045	ZU3EG	7Z020	ZCU102	ZU3EG	ZU9EG	ZU9EG	7VX690T	7A200T
DSP	109 (12%)	360 (10%)	202 (92%)	1144 (45%)	-	403 (16%)	1452 (58%)	1352 (36%)	215 (29%)
LUT	9K (41%)	52K (73%)	38K (71%)	552K (92%)	36K (51%)	38K (14%)	139K (51%)	390K (56%)	69K (38%)
BRAM	110 (20%)	159 (74%)	97 (69%)	912 (48%)	432 (100%)	142 (16%)	1729 (95%)	1460 (99%)	364 (100%)
External Memory	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
Power (W)	2.1	5.5	2.6	23.6	10.2	3.4	-	15.4	**1.4**
Clock (MHz)	100	250	250	200	220	-	150	200	100
Inf./sec (FPS)	1.4	41.1	20.5	446	200	75	127.4	987	141
Inference/J	0.6	7.5	7.9	18.9	19.6	22.1	-	64.1	**97.7**

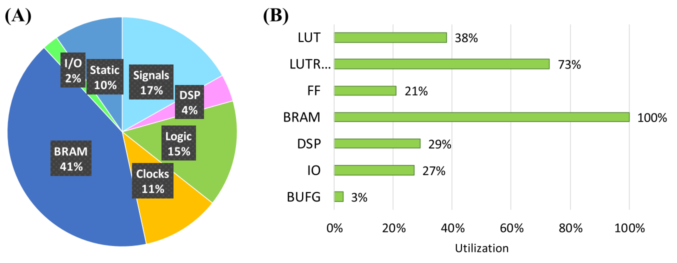


Fig. 14. Implementation results of the hardware with 16 PEs and 12 MAC units per PE on the Xilinx Artix 7A200T FPGA from the AC701 evaluation board. (a) Power breakdown. (b) Resource utilization.

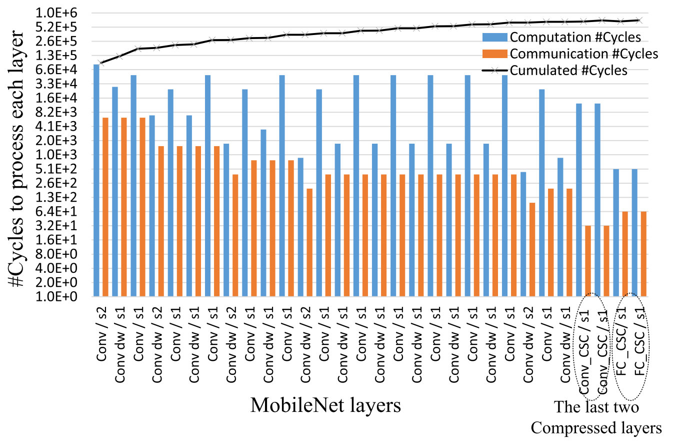


Fig. 15. Number of computation and communication cycles per layer implementation of the 0.5 CSC-MobileNet-192 as a workload, over the hardware with 16 PEs and 12 MAC units per PE.

model sparsified to 10.80% and 11.03% using fine-grained and coarse-grained pruning methods, respectively, our implementation has the same level of accuracy, yet more than 7× less power dissipation and more than 1.8× more energy efficiency. While most of the related works in Table VIII use heterogeneous computing platforms that employ FPGAs along with DRAMs and/or CPUs, our implementation is a minimal design that relies solely on the FPGA resources to implement a compact MobileNet and to enjoy efficiency advantages. In fact, the MobileNet in our work was compressed to the extent that a fully on-chip processing method on a small FPGA would be justified. This implementation style is also practiced in the work of Su *et al.* [42] for a redundancy-reduced MobileNet, thereby allowing their implementation to benefit from the high bandwidth of FPGA on-chip BRAMs and to gain a performance on par with ours (127 FPS versus 141 FPS). Similar works that deploy MobileNets to large FPGAs for full on-chip processing with the aim of gaining high throughput include works of [43]–[45] that report throughput of 1131, 3109, and 5157 FPS on Intel Stratix FPGAs. However, the power consumption and the efficiency metrics in these works are not of concern, hence not reported. While this implementation style is not always practical when large DCNN models are targeted for small FPGAs, it is worth emphasizing that the main reason for the efficiency outperformance in our implementation is the usage of a small DCNN that fits for full on-chip processing in a tiny FPGA (Artix-7), which is significantly smaller and an order of magnitude as less power dissipating as most of the FPGAs used in the related work discussed in this article.

IX. CONCLUSION

To address the general issues with pruning methods and compact model architectures, we introduced CSC architectures with a memory/computation complexity of $\mathcal{O}(N \log N)$ that can be used as an overlay for both FC and CONV layers of $\mathcal{O}(N^2)$, where N is the number of nodes/channels given a layer. CSC architectures can effectively compress both traditional FC and CONV layers in tandem with extreme quantization and can be implemented using a hardware-friendly style. Furthermore, both standard convolution and depthwise convolution layers conform to special cases of the CSC layers where the mathematical functions related to these layers can be merged into a single formulation and implemented indistinguishably under one arithmetic logic component. The effectiveness of the CSC architecture is examined for compression of popular 2–32-bit DCNNs. As a result, we showed that using the CSC architecture, the AlexNet can be compressed on par with pruning method. We also managed to further compress MobileNets up to ∼1.5× by replacing their last two layers with appropriate CSC layers. The compressed 0.5 CSC-MobileNet-192 can be easily deployed to low-power Artix-7 FPGA for full on-chip processing due to the small memory requirement. Compared to state of the art, our implementations are more than 1.5× less power consuming while more than 1.8× superior in terms of energy efficiency for the ImageNet classification when directed for FPGAs.

REFERENCES

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, “Hello edge: Keyword spotting on microcontrollers,” 2017, *arXiv:1711.07128*. [Online]. Available: http://arxiv.org/abs/1711.07128

[2] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[3] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[5] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[6] A. Mirzaeian, H. Homayoun, and A. Sasan, "NESTA: Hamming weight compression-based neural Proc. Engine," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 530–537.

[7] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 218–220.

[8] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[9] C. Deng, S. Liao, and B. Yuan, "PermCNN: Energy-efficient convolutional neural network hardware architecture with permuted diagonal structure," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 163–173, Feb. 2021.

[10] A. Shiri, B. Prakash, A. N. Mazumder, N. R. Waytowich, T. Oates, and T. Mohsenin, "An energy-efficient hardware accelerator for hierarchical deep reinforcement learning," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–4.

[11] S. Kala, B. R. Jose, J. Mathew, and S. Nalesh, "High-performance CNN accelerator on FPGA using unified winograd-GEMM architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2816–2828, Dec. 2019.

[12] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1953–1965, Sep. 2020.

[13] J. Liao, L. Cai, Y. Xu, and M. He, "Design of accelerator for MobileNet convolutional neural network based on FPGA," in *Proc. IEEE 4th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, vol. 1, Dec. 2019, pp. 1392–1396.

[14] J. Sanchez, A. Sawant, C. Neff, and H. Tabkhi, "AWARE-CNN: Automated workflow for application-aware real-time edge acceleration of CNNs," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9318–9329, Oct. 2020.

[15] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015.

[16] NVIDIA, "CUDA toolkit 4.2," NVIDIA, 2012.

[17] K. Alizadeh Vahid, A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient FFT based neural architecture design," 2019, *arXiv:1906.02256*. [Online]. Available: http://arxiv.org/abs/1906.02256

[18] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[19] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360

[20] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*. [Online]. Available: http://arxiv.org/abs/1405.3866

[21] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, "PermDNN: Efficient compressed DNN architecture with permuted diagonal matrices," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 189–202.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[23] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*. [Online]. Available: http://arxiv.org/abs/1605.04711

[24] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: http://arxiv.org/abs/1606.06160

[25] C. N. Coelho, Jr. *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," 2020, *arXiv:2006.10159*. [Online]. Available: http://arxiv.org/abs/2006.10159

[26] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang, "Building fast and compact convolutional neural networks for offline handwritten Chinese character recognition," *Pattern Recognit.*, vol. 72, pp. 72–81, Dec. 2017.

[27] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," 2016, *arXiv:1608.03665*. [Online]. Available: http://arxiv.org/abs/1608.03665

[28] T. Zhang *et al.*, "StructADMM: A systematic, high-efficiency framework of structured weight pruning for DNNs," 2018, *arXiv:1807.11091*. [Online]. Available: http://arxiv.org/abs/1807.11091

[29] L. Yang, Z. He, and D. Fan, "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 6623–6630.

[30] G. Srivastava, D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J.-S. Seo, "Joint optimization of quantization and structured sparsity for compressed deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 1393–1397.

[31] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, p. 297, May 1965.

[32] C. Ding *et al.*, "C ir CNN: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 395–408.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[34] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[35] H. Mao *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*. [Online]. Available: http://arxiv.org/abs/1705.08922

[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[37] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM SIGARCH Comput. Archit. News*, 2016, pp. 243–254.

[38] Y. Yang *et al.*, "Synetgy: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, pp. 23–32.

[39] Y. Chen *et al.*, "T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 13–18.

[40] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 17–25.

[41] M. Blott *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Dec. 2018.

[42] J. Su *et al.*, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification," in *Proc. Int. Symp. Appl. Reconfigurable Comput.* Cham, Switzerland: Springer, 2018, pp. 16–28.

[43] R. Zhao, H.-C. Ng, W. Luk, and X. Niu, "Towards efficient convolutional neural network for domain-specific applications on FPGA," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 147–1477.

[44] Y. Zhao *et al.*, "Automatic generation of multi-precision multi-arithmetic CNN accelerators for FPGAs," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2019, pp. 45–53.

[45] M. Hall and V. Betz, "From TensorFlow graphs to LUTs and wires: Automated sparse and physically aware CNN hardware generation," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2020, pp. 56–65.

[46] B. Prakash *et al.*, "Improving safety in reinforcement learning using model-based architectures and human intervention," in *Proc. 32nd Int. Flairs Conf.*, 2019.