Pre-Defined Sparsity for Low-Complexity Convolutional Neural Networks

Souvik Kundu, *Member, IEEE*, Mahdi Nazemi, *Member, IEEE*, Massoud Pedram, *Fellow, IEEE*, Keith M. Chugg, *Fellow, IEEE*, and Peter A. Beerel, *Senior Member, IEEE*

Abstract—The high energy cost of processing deep convolutional neural networks impedes their ubiquitous deployment in energy-constrained platforms such as embedded systems and IoT devices. This article introduces convolutional layers with pre-defined sparse 2D kernels that have support sets that repeat periodically within and across filters. Due to the efficient storage of our periodic sparse kernels, the parameter savings can translate into considerable improvements in energy efficiency due to reduced DRAM accesses, thus promising significant improvements in the trade-off between energy consumption and accuracy for both training and inference. To evaluate this approach, we performed experiments with two widely accepted datasets, CIFAR-10 and Tiny ImageNet in sparse variants of the ResNet18 and VGG16 architectures. Compared to baseline models, our proposed sparse variants require up to $\sim 82\%$ fewer model parameters with $5.6\times$ fewer FLOPs with negligible loss in accuracy for ResNet18 on CIFAR-10. For VGG16 trained on Tiny ImageNet, our approach requires $5.8\times$ fewer FLOPs and up to $\sim 83.3\%$ fewer model parameters with a drop in top-5 (top-1) accuracy of only 1.2% ($\sim 2.1\%$). We also compared the performance of our proposed architectures with that of ShuffleNet and MobileNetV2. Using similar hyperparameters and FLOPs, our ResNet18 variants yield an average accuracy improvement of $\sim 2.8\%$.

Index Terms—Convolutional neural network (CNN), pre-defined sparsity, parameter reduction, complexity reduction, energy-efficient CNN, storage aware sparsity

1 Introduction

In recent years, deep convolutional neural networks (CNNs) have become critical components in many real world vision applications ranging from object recognition [2], [3], [4], [5] and detection [6], [7], [8] to image segmentation [9]. With the demand for high classification accuracy, current state-of-the-art CNNs have evolved to have hundreds of layers [2], [3], [4], [10], [11], requiring millions of weights and billions of FLOPs. However, because a wide variety of neural network applications are heavily resource constrained, such as those for embedded and IoT devices, there is increasing interest in CNN architectures that balance implementation efficiency with accuracy and associated hardware accelerators that target CNNs [12], [13], [14]. In particular, because energy is often the primary limited resource, researchers have focused on minimizing the number of non-zero model parameters and the accelerator's access to off-chip DRAM, which consumes around 200× more energy than access to on-chip SRAM [15].

Previous work has focused on accelerating inference and proposed *model pruning* [16], [17], [18], [19], [20] and *quantization* [21], [22], [23], [24], [25], [26] to reduce the number of non-zero parameters. Recently, a more detailed analysis showed that such *unstructured pruning* may not reduce energy consumption because of the overhead required to manage sparse matrix representations [18]. This motivates *structured pruning* [18] which favors structure in the sparsity patterns that can more efficiently be managed in inference hardware.

Other work focused on the efficiency of both inference and training acceleration by defining notions of pre-defined sparsity [27], [28] in which a subset of the weights are fixed at zero before training and remain zero through inference. For example, a recent work [27] showed that neural networks can be trained with pre-defined hardware-friendly sparse connectivity in the fully connected multilayer perceptrons layers that avoids costly sparse matrix representations and thus can both speed-up and reduce the energy consumption of both inference and training. Other researchers have tried to address convolution (CONV) layers' computation complexity issue, which contribute the largest number of FLOPs for deep networks, exemplified by the CONV layer in ResNet18 [5] which accounts for $\sim 98\%$ of the total FLOPs for Tiny ImageNet classification. In particular many investigations have focused on efficient *pre-defined computationally-limited* filter designs to reduce complexity of training and inference at the cost of accuracy, including MobileNet [29], MobileNetV2 [30], and ShuffleNet [31].

This paper proposes pre-defined sparse convolutions to improve energy and storage efficiency during both training and inference. We refer to this approach as *pSConv* and presented initial simulation results that show negligible performance degradation compared to fully-connected baseline models in [1]. However, as mentioned earlier, unstructured forms of pSConv may not lead to energy reductions due to the overhead of managing their sparse matrix representations.

• *The authors are with Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089. E-mail: {souvikku, mnazemi, pedram, chugg, pabeerel}@usc.edu.*

Manuscript received 15 Oct. 2019; revised 27 Jan. 2020; accepted 31 Jan. 2020. Date of publication 10 Feb. 2020; date of current version 9 June 2020. (Corresponding author: Souvik Kundu.) Recommended for acceptance by N. Abu-Ghazaleh. Digital Object Identifier no. 10.1109/TC.2020.2972520

Motivated by this fact, we extend pSConv by proposing a form of periodicity, repeating a relatively small pattern of pre-defined sparse kernels within a 3D filter such that fixed zero-weights occur repeatedly with a constant interval across the 3D filter. *This periodicity can greatly reduce the overhead associated with managing sparsity, allowing the proposed CNN architecture to exhibit significant reductions in energy consumption compared to baseline CNNs with dense filters.*

Finally, we present a convolutional channel modification to boost the accuracy of pSConv-based CNNs. In particular, the accuracy loss incurred due to the added periodicity constraint may be non-negligible in some cases. To combat this phenomenon, we introduce fully-connected (FC) 2D kernels at fixed intervals within a 3D filter. *In particular, extending the periodic pattern of pre-defined sparse kernels with a fully connected kernel boosts accuracy while maintaining relatively low storage overhead.*

To evaluate the effectiveness of our proposed sparsity based CONVs, we run image classification tasks on variants of VGG [3] and ResNet [5] with CIFAR-10 [32] and Tiny ImageNet [33] datasets. We also show that we achieve higher test accuracy than MobileNetV2 [30] with similar network hyperparameter settings on these datasets. Finally, we analytically quantify the benefits of our algorithm compared to traditional approaches in terms of both FLOPs and storage, the latter assuming a variety of well-known sparse matrix representations.

The remainder of this paper is structured as follows. Section 2 provides notable related work in the domain of CNN architectures and efficient sparse matrix representations. Section 3 describes our proposed architecture in detail and is followed by our analytical evaluation of FLOPs and storage requirements in Section 4. We present our simulation results in Section 5 and conclude in Section 6.

2 Preliminaries and Related Work

CONV layers in neural network architectures transform the input images into abstract representations known as feature maps. To generate the output feature maps (OFMs) the filters of a layer are convolved with input feature maps (IFMs) which is comprised of the element wise product of

TABLE 1
Descriptions of Tensor Dimensions in a Convolutional Layer

Variable	Description
N	batch-size of a 3D feature map
H_i, W_i	height, width of IFM to a layer
H_f, W_f	height, width of a 2D kernel in a layer
H_o, W_o	height, width of OFM from a layer
C_i	# of IFM channels/# of 3D filter channels
C_o	# of OFM channels/# of 3D filters
C_g	# of channels in a group from GWC
n	# of parameters per kernel not pre-defined to be zero

filter and IFMs and the accumulation of partial sums. In particular, the following equation shows the computation of each OFM element in a standard fully-connected convolution (SFCC) layer.

$$O[z][v][x][y] = \mathrm{ReLU}\Bigg(B[v] + \sum_{k=0}^{C_i-1}\sum_{i=0}^{H_f-1}\sum_{j=0}^{W_f-1} I[z][k][Sx+i][Sy+j]W[v][k][i][j]\Bigg) \quad (1)$$
$$0 \le z < N, 0 \le v < C_o,$$
$$0 \le x < H_o, 0 \le y < W_o.$$

Here, O, I, W are the 4D OFM, IFM, and filter weight tensors, respectively and B is the 1D bias tensor added to each 3D filter result. Also, $O[z][v][x][y]$ represents the (x, y)th OFM element in the vth output channel corresponding to the input batch z. Note the extensive data reuse both in IFM and weights, for which optimized dataflow is needed to ensure energy efficiency [15], [34], [35]. The number of FLOPs necessary to generate the OFM for a SFCC layer can be estimated as

$$FL_{SFCC} = k^2 H_o W_o C_o C_i, \quad (2)$$

where, k represents both height (H_f) and width (W_f) of the 2D kernel and the meaning of the other variables are defined in Table 1. Also, in this paper we assumed stride size of 1 and consider a FLOP and a multiply-accumulate operation to be equivalent.

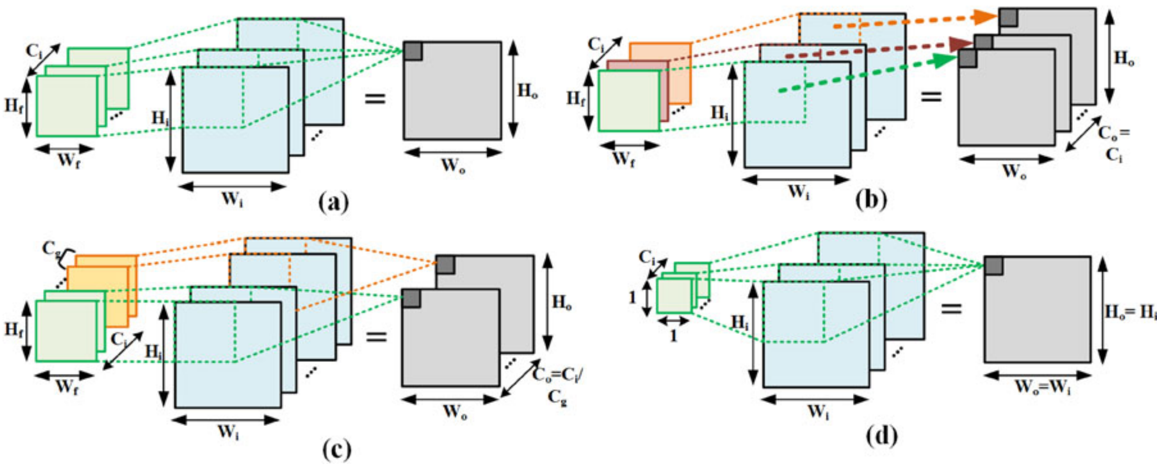


Fig. 1. Four major variants of convolutions: (a) standard fully connected convolution (SFCC), (b) depth-wise convolution (DWC), (c) group-wise convolution (GWC), and (d) point-wise convolution (PWC).

2.1 Pre-defined Computationally Limited Filters

Because the SFCC [36], shown in Fig. 1a, is computationally intensive, several pre-defined computationally-limited filters have been proposed to reduce the complexity of convolution. These filters can be broadly classified into three different categories, as shown in Fig. 1. The first category is depth-wise convolution (DWC) [37], shown in Fig. 1b. Here, each 2D kernel of size $H_f \times W_f$ is convolved with a single channel of the IFM to produce the corresponding OFM; thus C_i 2D kernels will produce an OFM of dimension $H_o \times W_o \times C_i$. This requires C_i times less computations compared to SFCC, but the output features capture no information *across* channels.

The second category is group-wise convolution (GWC) [2], shown in Fig. 1c, which provides a compromise between SFCC and DWC. Here, a single channel of the OFM is computed by convolving groups of C_G channels from the IFM with partitions of the 3D filters, each of size $H_f \times W_f \times C_G$. Thus, with a total number of groups $G = C_i/C_G$, a 3D filter of dimension $H_f \times W_f \times C_i$ provides an OFM of size $H_o \times W_o \times G$. Interestingly, SFCC can be viewed as GWC with $C_G = 1$ and DWC can be viewed as GWC with $C_G = C_i$. Typically, the number of groups G is chosen to be a small power of 2, but the choice is highly network architecture dependent [38].

Finally, Fig. 1d illustrates PWC in which the 2D kernel dimension has size 1×1, thus generating a single OFM channel with low complexity. In particular, compared to a 3×3 2D kernel dimension, the PWC has $9\times$ lower computational complexity. However, OFMs generated through this approach do not contain any embedded information *within* a channel.

Many well known network architectures have taken advantage of the benefits of pre-defined computationally-limited filters. For example, a combination of GWC and PWC was used in [38] and in the Inception modules [11], [39]. The ResNext architecture [40] also uses a combination of GWC and PWC to replace each CONV layer of ResNet [5]. A class of scaled-down, reduced parameter architectures that replace most of the 3×3 filters with PWC filters was dubbed SqueezeNet in [41]. MobileNet and MobileNetV2, two popular variants of low complexity architectures designed to be implemented in mobile devices, replace the SFCC layer with a DWC followed by a PWC layer to gather information *across* channels. ShuffleNet [31] uses a combination of GWC, a channel shuffling for information sharing across channels, followed by a DWC layer.

2.2 Sparse Matrix Storage Formats

Most hardware platforms that process deep neural networks can benefit from sparse weight matrices only when such weights are represented through sparse matrix storage formats. These formats typically store non-zero elements of a given matrix in a vector while auxiliary vectors describe the locations of non-zero elements. This section explains three such methods commonly employed.

2.2.1 Coordinate List (COO)

The COO format [42] uses three vectors to represent a sparse matrix: a *data* vector which keeps the values of non-zero elements, a *row* vector which stores the row indices of non-zero elements, and finally, a *column* vector which keeps track of column indices of non-zero elements. For example, consider the sparse matrix M shown below.

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 2 & 0 & 3 \\ 4 & 0 & 0 & 5 & 6 & 0 & 7 \\ 0 & 0 & 0 & 8 & 9 & 0 & 0 \end{bmatrix}.$$

The data, row, and column vectors for this matrix are as follows:

$$\text{data} = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9]$$

$$\text{row} = [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2]$$

$$\text{column} = [1 \quad 4 \quad 6 \quad 0 \quad 3 \quad 4 \quad 6 \quad 3 \quad 4].$$

In this representation, the size of all three vectors are the same and equal to the number of non-zero elements in the original sparse matrix.

2.2.2 Compressed Sparse Row (CSR)

Similar to the COO format, the CSR format [42] uses three vectors to represent a sparse matrix. The *data* vector stores values of non-zero elements in the order they are encountered when traversing the elements of the original matrix from left to right and top to bottom. The *column* vector keeps track of the column indices of non-zero elements, and the *index* vector stores additional information used to identify the indices of the elements of each row of the matrix within the data vector. In fact, the column vector is the same as the one in the COO while the index vector stores the row vector in the COO format in a compressed manner, hence the name CSR. As an example, the data vector and the auxiliary vectors for the sparse matrix M are as follows:

$$\text{data} = [\mathbf{1} \quad 2 \quad 3 \quad \mathbf{4} \quad 5 \quad 6 \quad 7 \quad \mathbf{8} \quad 9]$$

$$\text{column} = [1 \quad 4 \quad 6 \quad 0 \quad 3 \quad 4 \quad 6 \quad 3 \quad 4]$$

$$\text{index} = [0 \quad 3 \quad 7 \quad 9].$$

Here, the bold entries in the data vector indicate the first nonzero elements of a new row of M and occur at indices 0, 3, and 7 in the data vector. Thus, storing these indices in the index vector, along with the column vector, determine both the row and column for each element of the data vector. The index vector always begins with zero and ends with the length of the data vector. If a row of M has no nonzero elements, the corresponding element in the index vector is repeated.

2.2.3 Compressed Sparse Column (CSC)

The CSC format [42] is very similar to the CSR and, in fact, is equivalent to CSR storage of the transpose of M. The column vector for CSR storage of M^T is the *row* vector for CSC storage of M as follows

$$\text{data} = [\mathbf{4} \quad \mathbf{1} \quad \mathbf{5} \quad 8 \quad \mathbf{2} \quad 6 \quad 9 \quad \mathbf{3} \quad 7]$$

$$\text{row} = [1 \quad 0 \quad 1 \quad 2 \quad 0 \quad 1 \quad 2 \quad 0 \quad 1]$$

$$\text{index} = [0 \quad 1 \quad 2 \quad 2 \quad 4 \quad 7 \quad 7 \quad 9].$$

Similar to the CSR format, in the CSC format, the size of the data and row vectors are the same and equal to the number of non-zero elements. However, the size of the index vector is equal to the number of columns in the sparse matrix plus one.

Some of the existing deep neural network (inference) accelerators such as Cambricon-X [43], Eyeriss [13],[1] and Eyeriss v2 [14] have hardware support for processing values represented using sparse storage formats. The periodic sparsity introduced in this work allows us to further compress sparse representations such as the CSR and CSC formats by reusing the auxiliary vectors. This not only decreases the storage required for keeping model parameters in memory but also reduces the energy associated with transferring them from the main memory to processing elements (PEs). Furthermore, the proposed optimized sparse storage formats can be integrated into some of the existing accelerators such as Eyeriss v2 with minor modifications to the controller logic or PEs. Section 4 details the storage and energy savings achieved through deployment of the proposed formats.

3 Pre-Defined Sparsity

This section first describes pSConv, a form of pre-defined sparse kernel based convolution that we initially proposed in [1]. It then describe how we introduce periodicity to this framework to reduce the overhead of managing sparse matrix representations. Finally, the section presents a method to boost accuracy by periodically introducing a fully connected kernel into the 3D filters.

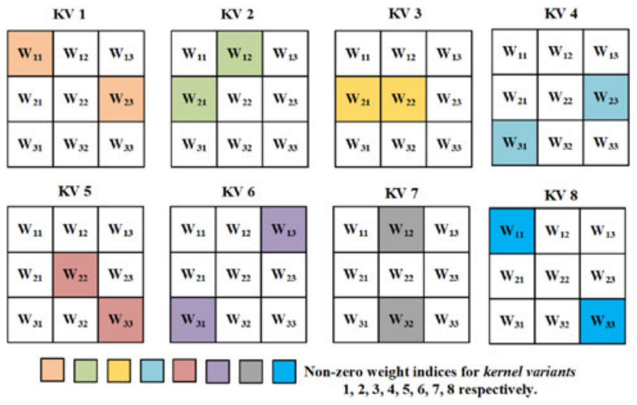
We define the *kernel support* as the set of entries in a $k \times k$ 2D kernel that are not constrained to be zero. The size of this set is defined as *kernel support size (KSS)*. The *kernel variant size* (KVS) is defined as the number of kernels with unique kernel support in a 3D filter.

3.1 Pre-Defined Sparse Kernels

We say a 3D filter of size $k \times k \times C_i$ has pre-defined sparsity if some of the $k^2 \times C_i$ parameters are fixed to be zero before training and held fixed throughout training and inference. A *regular* pre-defined sparse 3D filter has the same KSS for each kernel that comprises the 3D filter.[2] This regularity can help reduce the workload imbalance across different PEs performing multiply-accumulates and thus can help improve throughput of CNN accelerators [14]. Fig. 2 shows an example of kernel variants. Here, $k = 3$, meaning $KSS = 9$ denotes the standard kernel without any pre-defined sparsity and $KSS = 2$ signifies that seven of the nine kernel entries are fixed at zero. The choice of kernel variants can be viewed as a model search problem, however, in this paper we adopted a lower complexity approach of choosing them in a constrained pseudo-random manner which ensures every possible locations in k^2 2D kernel space (9 in this case) has at-least one entry in a 3D filter which is not pre-defined to be zero. As an example, Fig. 3 illustrates

1. Note that Eyeriss [13] uses run-length coding (RLC) to represent sparse vectors, in particular sparse activations, whereas CSR and CSC are better suited to represent sparse matrices.

2. We only consider the convolutional weights when defining sparsity. Bias and other variables associated with batchnorm are not considered because they add negligible complexity.



KV 1
KV 2
KV 3
KV 4
KV 5
KV 6
KV 7
KV 8
W11 W12 W13
W21 W22 W23
W31 W32 W33
Non-zero weight indices for kernel variants 1, 2, 3, 4, 5, 6, 7, 8 respectively.

Fig. 2. An example of pre-defined sparse kernels with eight different kernel variants each having KSS of two. The colored locations in each 2D kernel are allowed to have non-zero weight values.

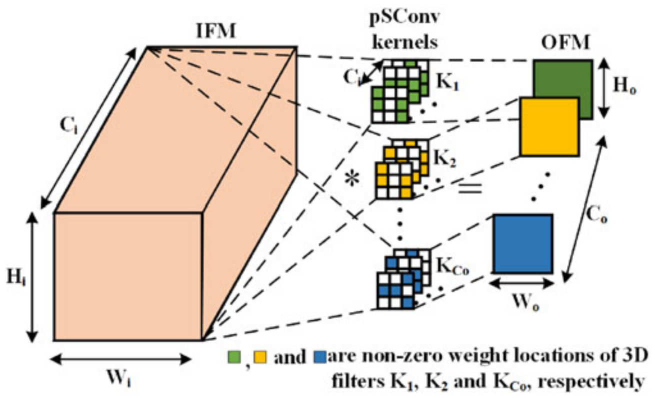
how an OFM of size $H_o \times W_o \times C_o$ is generated through convolution of $C_i \times C_o$ pre-defined sparse kernels of size k^2 with an IFM of size $H_i \times W_i \times C_i$.

The challenge with efficiently implementing this scheme is how to avoid processing the weight entries that are fixed at zero. Because the *kernel variants* are chosen randomly from a potential set of $\binom{k^2}{KSS}$ options and KVS could be as large as C_i for each 3D filter, the non-zero weight index memories can represent considerable overhead. We propose to address this problem by introducing periodicity within a 3D filter, as described below.

3.2 Periodic Sparse Kernel Patterns

In order to reduce the overhead of storing the sparsity patterns, we propose to repeat the sparsity patterns, using only a small number of kernel variants across all filters. This is particularly beneficial in the compressed sparse weight formats because the same index values can be used for multiple filters.

Fig. 4 shows an example of periodically repeating kernel patterns, with a periodicity $P = KVS = 4$. Notice to retain periodicity across different 3D filters and while still providing some diversity, we rotate the sequence of kernel variants, starting each filter (of P consecutive filters) with a different kernel variant. For instance, if the first 3D filter starts with KV1 followed by KV2, KV3, and KV4, and then repeats the order, we start the second 3D filter with KV2 to create a



IFM
pSConv kernels
OFM
C_i
K_1
K_2
K_{Co}
H_o
C_o
W_o
H_i
W_i
*
=
🟩, 🟨 and 🟦 are non-zero weight locations of 3D filters K_1, K_2 and K_{Co}, respectively

Fig. 3. An example of proposed pre-defined sparse kernel based convolution with KSS of 4.

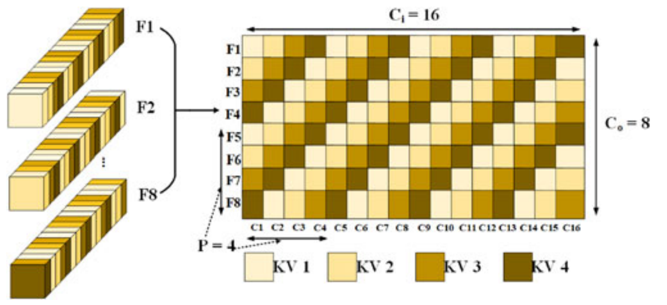


Fig. 4. Regular sparse kernel based 4D weight tensor. In the figure the 4D weight tensor has four different types of 2D kernel i.e., four different KVs (colored differently).

repeating sequence of [KV2, KV3, KV4, KV1]. Thus, we maintain the sequence of repeating kernels modulo rotation.

Our specific choice of sparse KVs in our experiments are obtained by sequentially picking non-zero 2D entries randomly constrained such that no non-zero 2D entry is chosen twice until all entries of the kernel are chosen at least once. Furthermore, we ensure that every pixel in the input frame has an opportunity to affect the outcome of our sparse-periodic network which constrains the minimum value of periodicity P. For example, for a 3×3 kernel, with KSS of 1, the minimum value of P necessary to ensure every entry of the kernel is chosen is 9. More specifically, the nine sparse 3×3 2D kernels in this example each must have a different single non-zero entry such that together they cover all entries.

3.3 Boosting Accuracy With FC Kernels

Although the periodicity in sparse patterns is beneficial for overhead management of the sparsity, the choice of KSS and the simplistic way of choosing *kernel variants* may sometimes cost significant classification performance. Methods to find suitable sparse patterns and KVS values through pattern pruning, inspired by image smoothing filters, were recently considered in [44]. However, here, we propose a complementary approach in which we periodically introduce fully connected kernels, i.e., kernels with KSS = k^2, within each 3D filter. We use η to denote the number of dense or FC kernels in a period P and, in principle, it can have any value between 0 and P. In the case of standard (dense) convolution filter based models, $\eta = P$. In contrast, $\eta = 0$ implies no boosting. Fig. 5 illustrates the idea where one fully connected kernel (η = 1) is introduced every P kernels and with other $P-1$ sparse kernels, repeating this P pattern throughout the 3D filter.

Note that our selection of the period P is premised on the fact that balancing the computational requirements across 3D filters is preferred for hardware implementations because it enables more efficient scheduling across parallel computational units. It is therefore desirable to have a fixed number of non-zero weights per filter which implies having an equal number of FC kernels per filter. Given the approach illustrated in Fig. 4 it is therefore preferred to have C_i be divisible by P. In our experiments, detailed in Section 5, the layers where boosting is applied have $C_i \in \{64, 128, 256, 512\}$. Thus our preferred values of P are {2, 4, 8, 16, 32, 64}.

To choose the sparse kernel variants we follow the same principle as described in Section 3.2 before adding the η FC kernels. However, in the presence of an FC kernel, P can, in principle, be lower than the minimum P without boosting because the FC kernel covers all entries. Moreover, when $KVS < P$, we propose to randomly reuse some of the sparse kernel variants to maintain periodicity.

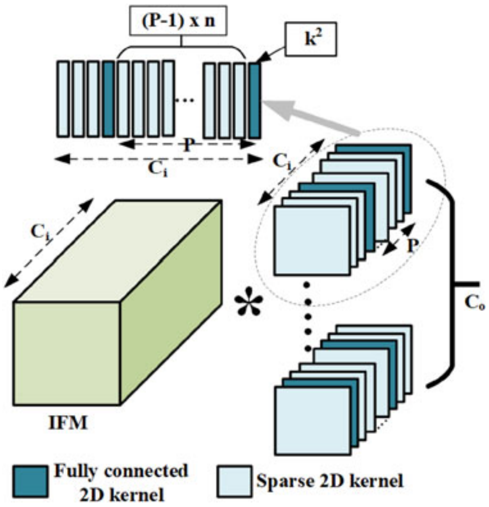


Fig. 5. Periodic insertion of FC 2D kernels between sparse kernels.

TABLE 2
Expression of FLOPs Count for Inference Operation With Various Pre-Defined Computationally-Limited Filters

Approach	FLOP count (forward, ideal)
MobileNet-like [29] (DWC+PWC)	$H_oW_oC_i(k^2 + C_o)$
ShuffleNet-like [31] (GWC+PWC)	$H_oW_oC_oC_i(\frac{k^2}{G} + 1)$

4 FLOPs and Energy Efficiency Analysis

4.1 Complexity Analysis

The total FLOPs for MobileNet-like and ShuffleNet-like CONV layers can be estimated as shown in Table 2. The total FLOPs for sparse (both periodic and aperiodic variants) kernel based CONV layers with KSS of n can be estimated as

$$FL_S = H_oW_oC_iC_on. \tag{3}$$

To estimate the FLOPs of sparse kernel based CONVs with boosting,[3] we start with the number of elements in a period (P) that are allowed to be non zero which can be computed as (shown in Fig. 5),

$$W_P = (P-1)n + k^2. \tag{4}$$

Now, with total number of (C_i/P) dense, and $(C_i - C_i/P)$ sparse kernels in each 3D filter of a layer the FLOPs can be computed as,

$$FL_{\text{PSD}} = \left[\left(\frac{C_i}{P}\right)k^2 + \left(C_i - \left(\frac{C_i}{P}\right)\right)n\right]H_oW_oC_o. \tag{5}$$

3. Here each period is assumed to have only one FC kernel.

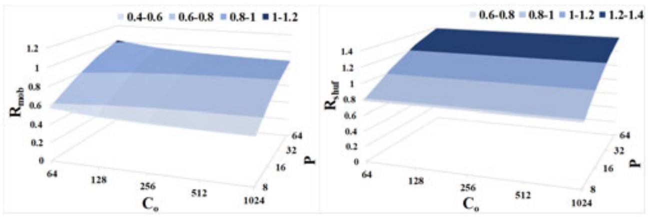


Fig. 6. A 3D illustration of the change in R_{mob} and R_{shuf} as a function of the C_o and P. Here we assumed G, k, and n to be 16, 3, and 1, respectively.

The ratio of the FLOP counts for MobileNet-like and ShuffleNet-like layers to that of sparse kernel based CONVs with boosting is

$$\begin{aligned} R_{\text{mob}} &= \frac{\text{FLOPs for MobileNet-like}}{\text{FLOPS for periodic-sparse with boosting}} \\ &= \frac{P(k^2 + C_o)}{[k^2 + (P-1)n]C_o} \end{aligned} \tag{6}$$

$$\begin{aligned} R_{\text{shuf}} &= \frac{\text{FLOPs for ShuffleNet-like}}{\text{FLOPS for periodic-sparse with boosting}} \\ &= \frac{P(\frac{k^2}{G} + 1)}{[k^2 + (P-1)n]}. \end{aligned} \tag{7}$$

It is clear that we will have computational saving when the values of R_{mob} and R_{shuf} are greater than 1. When C_o is large and $P >> \frac{k^2}{n}$, (6) and (7) can be approximated as

$$R_{\text{mob}} \simeq \frac{1}{n} \tag{8}$$

$$R_{\text{shuf}} \simeq \frac{(\frac{k^2}{G} + 1)}{n}, \tag{9}$$

which shows the complexity increment due to periodic insertion of FC kernels is negligible for relatively wide networks with large periods. Fig. 6 shows a 3D illustration of the per layer FLOP ratios (R_{mob} and R_{shuf}) as a function of C_o and P. Note that even though the per layer ratio can be less than 1, the total parameter count for MobileNet or ShuffleNet-like networks can be larger due to the presence of more layers.

4.2 The Impact of Periodicity on Storage and Energy

Sparsity leads to savings in storage only when the overhead of storing the auxiliary vectors to manage sparsity is negligible. This section presents a new sparse representation specifically tailored to periodic sparse kernels and compares it with existing formats. It also analyzes storage requirements of different sparse representations analytically, allowing the study of the effectiveness of such formats at different levels of density. Furthermore, it explains how the proposed representation can be exploited in CNN accelerators.

4.2.1 CSR/CSC With a Periodic Column/Row Vector

The periodic pattern of kernels introduced in Section 3.2 allows reusing the column/row vector in the CSR/CSC format. For example, assume a convolutional layer with 3×3 kernels, 128 input channels, 128 output channels, and a period of four. The 4D weight tensor corresponding to this convolutional layer can be represented by a *flattened weight matrix* where each row corresponds to a flattened filter. As a result, the number of rows in the flattened weight matrix is equal to 128 while the number of columns is $3 \times 3 \times 128 = 1152$. Because of the periodicity across filters, the structure of the rows of the flattened weight matrix will also repeat with a period of four. Therefore, one can simply store the column vector of the CSR format for the first four rows and reuse them for the subsequent rows. We refer to this new sparse storage format as CSR with a periodic column vector and denote it with CSR_P, where P denotes the period of repetition of the column vector.

Similarly, because of the periodicity of kernels within a filter, the columns of the flattened matrix also repeat with a period of $4 \times (3 \times 3) = 36$. As a result, one can choose to use the CSC format to represent the flattened sparse matrix and reuse the row vector for groups of 36 columns. We refer to this new format as CSC with a periodic row vector and denote it with CSC_P, where the P here denotes the period of repetition of the row vector.

Table 3 summarizes the notation used for comparing the storage cost of different storage formats. Using the notation introduced here, Table 4 explains storage requirements of different storage formats.

Based on Table 4, the COO format is expected to have higher overhead than that of the CSR and CSC formats, which have similar storage overhead. Furthermore, it is evident that the introduction of periodicity to the CSR and CSC formats can significantly decrease the storage overhead.

TABLE 3
Summary of Notation for Matrix Storage Formats

Variable	Description
H_F, W_F	height, width of a flattened weight matrix
ρ	density ($0 \leq \rho \leq 1$)
b_v	number of bits for representing data values
b_r, b_c	number of bits for representing row, column values
b_i	number of bits for representing index values
b_P	number of bits for representing the period

4.2.2 Application to Weight Sub-Matrices

As noted above, a convolutional layer with periodic sparse kernels induces a flattened weight matrix that also has periodically repeating columns and rows. In a CNN accelerator, the processing of a convolutional layer is often broken down into smaller operations where subsets of the flattened weight

TABLE 4
Storage Requirement of Storing a Matrix Using Dense and Sparse Storage Formats

Format	Storage Requirement (bits)
Dense	$H_F W_F b_v$
COO	$\rho H_F W_F (b_v + b_r + b_c)$
CSR	$\rho H_F W_F (b_v + b_c) + (H_F + 1) b_i$
CSC	$\rho H_F W_F (b_v + b_r) + (W_F + 1) b_i$
CSR_P	$\rho H_F W_F b_v + \rho P W_F b_c + (H_F + 1) b_i + b_P$
CSC_P	$\rho H_F W_F b_v + \rho P H_F b_r + (W_F + 1) b_i + b_P$

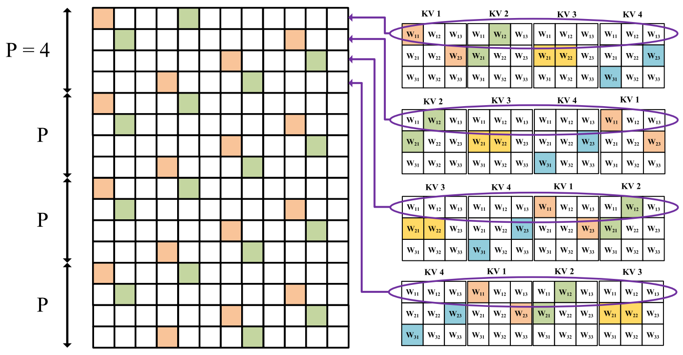


Fig. 7. Illustration of how periodicity in a filter leads to repeating rows of sub-matrices of the filter's flattened weight matrix.

matrix are processed across multiple PEs. This processing requires accessing a sub-matrix of the flattened weight matrix. If this sub-matrix is large enough, it will also have row or column vectors that are repeated periodically. For example, Fig. 7 demonstrates a subset of a flattened weight matrix that is used in a single processing element of an architecture like Eyeriss v2 [14] (the original flattened weight matrix is built using the first four kernel variants shown in Fig. 2). This sub-matrix corresponds to processing the first (top) row of four kernels of 16 filters. Specifically, the sub-matrix consists of 16 rows corresponding to 16 filters and 12 columns corresponding to the top row of four kernels per filter. Note in Fig. 7, the four kernels have been rotated as described in Section 3.2. Based on the periodic pattern across filters, the sub-matrix shown in Fig. 7 has repeating rows with a period of four and can be represented using CSR_4.

Because each PE in a CNN accelerator processes a small portion of the flattened weight matrix, b_c, b_r, and b_i have small ranges and therefore can be represented using a small number of bits. For example, assuming $b_v = 8$, $b_c = b_r = 4$, and $b_i = 7$, Fig. 8a compares storage requirements of various existing storage formats at different levels of filter density. It is observed that the CSR and CSC formats yield lower total storage when the original matrix is at most 62 and 65% dense, respectively.

Fig. 8b compares storage requirement of dense, CSR, and $\mathrm{CSR_P}$ formats for the same matrix that was shown in Fig. 8a, for different values of P, and $b_P = 6$. It is observed that the CSR_8 and CSR_{16} yield lower total storage when the original matrix is at most 82 and 73% dense, respectively. Furthermore, at 62% density, CSR_8 and CSR_{16} yield lower total storage compared to CSR by 23 and 16%, respectively.[4] This is equivalent to 60.04 and 39.86% reduction in the overhead of storing auxiliary vectors for the CSR_8 and CSR_{16} compared to the CSR format, respectively.

Because the energy cost associated with transferring from the DRAMs is well-modeled as proportional to the number of bits read [45], the reduced storage requirements of $\mathrm{CSR_P}/\mathrm{CSC_P}$ lead to a proportional reduction in the energy cost associated with DRAM access. For example, a 50% savings in storage will result in a $\sim 2\times$ reduction in energy consumption related to DRAM access. For this reason, in the remainder of this paper, we focus on savings in storage requirements with the energy savings being implicit.

4. Interestingly, CSR has similar storage requirements as RLC. In particular, as implemented in Eyeriss [13], at 62% density, RLC would lead to 0.14% more storage than CSR.

4.2.3 Hardware Support for Periodic Sparsity

The low-complexity storage formats introduced in Section 4.2.1, i.e., $\mathrm{CSR_P}/\mathrm{CSC_P}$, cannot be integrated into existing accelerators without ensuring they can support the proposed periodic sparse format. For example, in Eyeriss v2, each weight value (i.e., data) is coupled with its corresponding index and they are read as a whole from the main memory. On the other hand, the $\mathrm{CSR_P}/\mathrm{CSC_P}$ store the column/row vector separately from the data vector and read the auxiliary vectors once for all data values. This not only requires proper adjustment of the bus that transfers data from the DRAM to the chip but also may require a minor modification in either the control logic or PEs.

One approach to make an accelerator like Eyeriss v2 compatible with periodic sparsity is to store the weights in DRAM using the proposed sparse periodic format and modify the system-level control logic to expand the column/row vector before storing them in the PE's scratchpad memory. In other words, the sparse column/row vector is read from the DRAM only once, but replicated before being written into the scratchpad memory corresponding the the column/row vector so that they adhere to the CSR/CSC format. In this manner, the scratchpad memory within each PE remains the same and stores bundled (data, index) pairs. Because DRAM accesses consume two orders of magnitude more energy than on-chip communication, we can thus achieve close to the optimal energy savings without requiring any change in the PE array or its associated control structures.

A more comprehensive approach to supporting periodic sparsity involves ensuring the PEs can use the column/row scratchpad memory as a configurable circular buffer, which, to support periodicity, will be configured to have length P. This type of support may already exist because in many cases, the size of the weight matrix processed within each PE is smaller than the size of the corresponding scratchpad memory and therefore, only a portion of the scratchpad memory is used. In this approach, the periodic column/row vector is read from the DRAM once, written into the scratchpad memory, and accessed multiple times for different rows of the weight matrix. This reduces the required on-chip communication and thus may save more memory compared to storing the expanded column/row vectors in the scratchpad memory.

While the presented approaches enable compression of the column/row vectors, one may be able to compress the index vector as well, as suggested by the row periodicity illustrated in Fig. 7. However, this may require more complex hardware support to expand the index vector before storing them in the PEs or adding support for the compressed index vectors within the PE.

5 Experimental Results

This section describes our simulation results and analysis. We first detail the datasets, architecture, and important hyperparameters used for our experiments, followed by our experimental results of our proposed pSConv approach, the introduction of periodicity, and our performance boosting

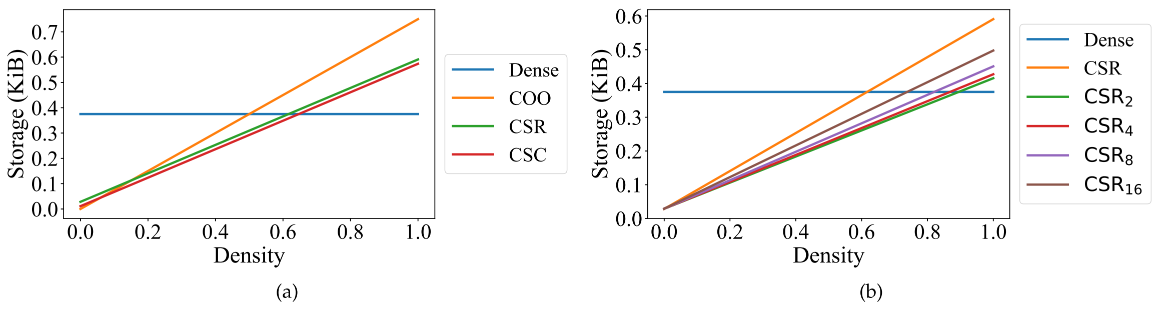


Fig. 8. Comparison of storage requirements of (a) various existing storage formats and (b) dense, CSR, and $\mathrm{CSR_P}$ formats at different levels of density for a matrix of size 32×12 ($b_v = 8$, $b_c = b_r = 4$, $b_i = 7$, and $b_P = 6$).

technique. Finally, we compare our modified network architectures with MobileNetV2 [30], a popular low-complexity CNN variant for image classification, in terms of FLOPs, model parameters, and accuracy. We used Pytorch [46] to design the models and trained/tested the models on AWS EC2 P3.2x large instances that have an NVIDIA Tesla V100 GPU.

TABLE 5
Nomenclature of the Network Architectures Used in Simulation

Name	Description of the network architecture
aaa_pSC<n>	aaa network with pre-defined sparse kernel based convolution where each 2D kernel has n weights not pre-defined to be zero.
aaa_pSC<n>_P<m>	aaa network with every mth kernel is FC and rest are pre-defined sparse kernels having n weights not pre-defined to be zero.
aaa_PS<n>_P<m>	aaa network with both periodicity and kernel variant values of m, and each 2D kernel has n weights not pre-defined to be zero.
aaa_PSD<n>_P<m>	aaa network with periodic kernel variants having periodicity m, where each period has $m-1$ sparse kernel variants each with n weights not pre-defined to be zero and 1 FC $k \times k$ kernel.

TABLE 6
Test Accuracy of pSConv Based VGG16, and ResNet18 on CIFAR-10 and Tiny ImageNet

Data set	Model	Top 1 acc (%)	Top 5 acc (%)	Parameters	Parameters (%) reduction
	VGG16_pSC9	**92.8**	–	14.73 M	—
C	VGG16_pSC4	92.0	–	6.55 M	55.56
I	VGG16_pSC2	91.2	–	3.27 M	77.78
F	VGG16_pSC1	89.5	–	1.64 M	88.89
A	ResNet18_pSC9	**92.9**	–	11.17 M	—
R	ResNet18_pSC4	92.5	–	5.06 M	54.65
10	ResNet18_pSC2	91.1	–	2.62 M	76.56
	ResNet18_pSC1	89.4	–	1.39 M	87.50
	VGG16_pSC9	**57.2**	78.9	14.73 M	—
	VGG16_pSC4	56.1	**79.1**	6.55 M	55.56
Tiny	VGG16_pSC2	54.2	78.2	3.27 M	77.78
Image	VGG16_pSC1	52.5	76.7	1.64 M	88.89
Net	ResNet18_pSC9	**62.4**	**83.2**	11.17 M	—
	ResNet18_pSC4	61.7	83	5.06 M	54.65
	ResNet18_pSC2	60.2	82.7	2.62 M	76.56
	ResNet18_pSC1	59.0	82.2	1.39 M	87.50

Here we use KSS of 9, 4, 2, and 1, respectively. Also, KSS of 9 means SFCC based CONVs and thus they are used as baseline to compare accuracy, and parameters.

5.1 Datasets, Architectures, and Hyperparameters

To evaluate our models we used CIFAR-10 [32] and Tiny ImageNet [33], two widely popular image classification datasets. The input image dimensions of CIFAR-10 and Tiny ImageNet are $(32 \times 32 \times 3)$ and $(64 \times 64 \times 3)$, respectively. The number of different output classes for these two datasets are 10 and 200, respectively. We chose variants of VGG16 [3] and ResNet18 [5] as the base network models to apply our architectural modifications. The VGG16 architecture has thirteen 3×3 kernel based convolutional layers. The flattened output of final CONV layer is fed to the fully connected part having three fully connected (FC) layers.[5] The CONVs of ResNet18 architecture consists of four layers each containing two basic blocks, where each basic block has two convolutional layers along with a skip connection path. We used pre-defined sparse kernels on all $k \times k$ CONV layers where $k > 1$ but excluded the first layer, as it is connected to the primary inputs and is thus more sensitive to zero weights. Training was performed for 120 and 100 epochs for CIFAR-10 and Tiny ImageNet, respectively. The initial learning rate was set to 0.1 with momentum of 0.9 and weight decay value to 5×10^{-4}. The image datasets were augmented through random cropping and horizontal flips before being fed into the network in batches of 128 and 100 for CIFAR-10 and Tiny ImageNet, respectively. All results reported are the average over two training experiments. Table 5 provides the names of each variant of network model and corresponding architecture descriptions.

5.2 Results for pSConv Based CNN

We analyzed three different variants of regular sparse kernel based CONVs with KSS values of 4, 2 and 1 along side the baseline standard convolution based network. As stated earlier, in our choice of kernel patterns we ensure each of the k^2 possible kernel entries are covered by at least one sparse kernel variant. Table 6 provides the results in terms of accuracy and parameter count[6] with the KSS variants applied in VGG16 and ResNet18 architectures. The ResNet18-based results show that even with KSS of only 4,

5. In VGG16 for CIFAR-10 dataset, we used only one FC layer because the input image dimension is $4\times$ smaller than Tiny ImageNet and multiple FC layers are not needed to achieve high accuracy.

6. We considered the convolution layer parameters only to report in the tables of this section without considering the overhead of indexing.

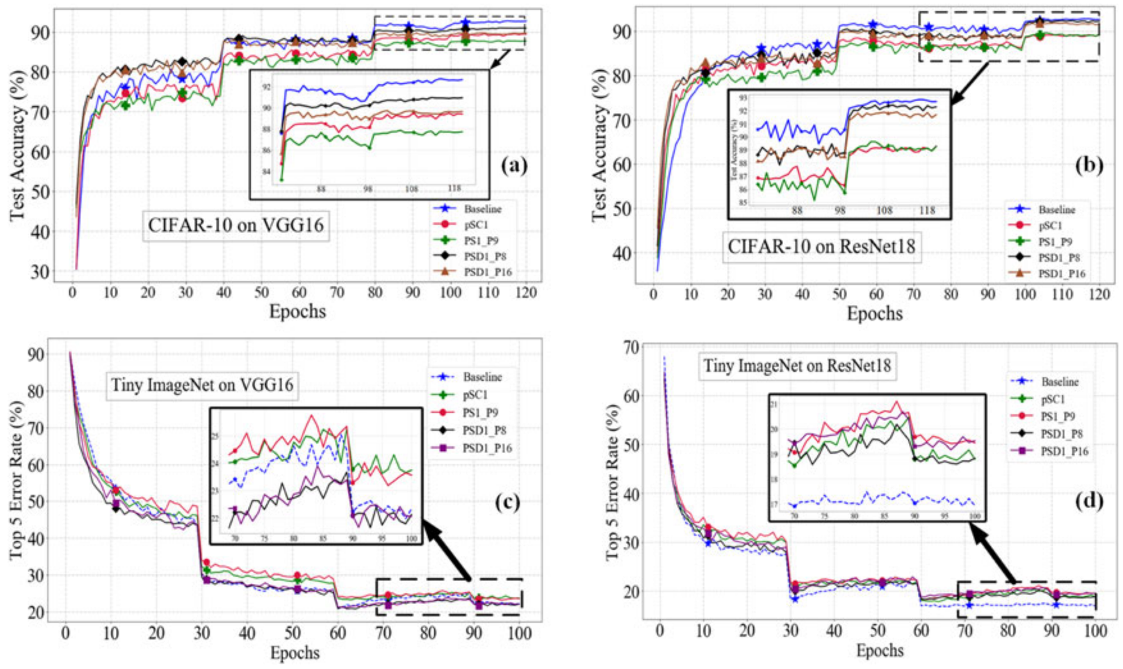


Fig. 9. (a), and (b) shows the test accuracy versus epochs for CIFAR-10 dataset in different variants of VGG16 and ResNet18 models, respectively; (c), and (d) are plots of top 5 error rate versus epochs for Tiny ImageNet dataset in different variants of VGG16 and ResNet18 models, respectively. The KSS for all the variants is 1.

the test accuracy degradation is within $\sim 0.4\%$ for CIFAR-10 dataset, and within $\sim 0.6\%$ for Tiny ImageNet. The same results for VGG16 show a test accuracy degradation is within $\sim 0.7\%$ for CIFAR-10 dataset, and within $\sim 1.1\%$ for Tiny ImageNet.

5.3 Results for pSConv With Periodicity

The storage and energy advantage associated with periodically repeating kernels with some specific set of kernel variants, analysed in Section 4.2, motivated us to evaluate its performance in terms of test accuracy. We leveraged the observation provided by [44] and kept the KVS $= P$ small for different KSS based architectures. In particular, as KSS of 4 covers more kernel entries per variant, we chose a corresponding P = KVS = 4 and covered all possible kernel entries of the 3×3 kernels. For similar reasons, we chose larger KVS for KSS of 2 and 1, respectively (6 and 9, respectively). We selected kernel variants as described in Section 5.2. Fig. 9 shows the learning curves for CIFAR-10 and Tiny ImageNet datasets with different variants of VGG16 and ResNet18 models with KSS of 1.[7] It is clear that the sparse variants learn at similar rates as the corresponding baselines.

TABLE 7
Test Accuracy of Different Variants of Periodic Sparse Kernel Based VGG16 and ResNet18 on CIFAR-10 and Tiny ImageNet

Data set	Model	(KVS, P)	Top 1 acc (%)	Top 5 acc (%)	Parameters	Parameter reduction (%)
C	VGG16_PS4_P4	(4, 4)	**91.7**	–	6.55 M	55.56
I	VGG16_PS2_P6	(6, 6)	90.6	–	3.27 M	77.78
F	VGG16_PS1_P9	(9, 9)	87.9	–	1.64 M	88.89
A	ResNet18_PS4_P4	(4, 4)	**92.9**	–	5.06 M	54.65
R	ResNet18_PS2_P6	(6, 6)	91.5	–	2.62 M	76.56
10	ResNet18_PS1_P9	(9, 9)	89.6	–	1.39 M	87.50
	VGG16_PS4_P4	(4, 4)	**56.9**	**79.9**	6.55 M	55.56
Tiny	VGG16_PS2_P6	(6, 6)	53.9	77.8	3.27 M	77.78
Image	VGG16_PS1_P9	(9, 9)	51.8	76.7	1.64 M	88.89
Net	ResNet18_PS4_P4	(4, 4)	**61.9**	**83**	5.06 M	54.65
	ResNet18_PS2_P6	(6, 6)	60.7	82.9	2.62 M	76.56
	ResNet18_PS1_P9	(9, 9)	58.9	81.8	1.39 M	87.50

The baseline architectures of Table 6 are used as the reference for calculating the reduction in parameters.

Table 7 shows the impact of an added periodicity constraint on test accuracy with our proposed variants. Note that because of the overhead of storing auxiliary vectors, the overall storage reduction is smaller than the ones reported in Table 7. For example, for VGG16_PS4_P4, the reduction in the number of parameters is 55.6%, but including the storage of the auxiliary vectors in CSR_4 format, the reduction is approximately 44.6%. If CSR format is used, the reduction in overall storage requirements, relative to the baseline is approximately 25%.

5.4 Results for Boosting

The results without and with periodically repeating sparse kernel patterns discussed in Sections 5.2 and 5.3, respectively, show considerable performance degradation at low KSS values such as 1. This section presents the performance of the network models with the proposed boosting method in which we periodically incorporate FC kernels ($k \times k$) in the 3D filter.[8]

To evaluate the value of boosting, we measure its impact when periodicity P is set to 8 and 16 as well as when applied

7. We saw similar trends with KSS of 2 and 4 in VGG16 and ResNet18, and so did not show in separate plots for brevity's sake.

8. In this paper, we focus on results with one FC kernel per period, i.e., $\eta = 1$. However, we also evaluated performance with larger values of η. For example, $\eta = 2$ for $P = 16$, yields similar accuracy as $\eta = 1$ for $P = 8$. Both models have similar parameter counts but the latter has significantly lower storage costs, suggesting restricting our model space to have $\eta = 1$ is reasonable.

TABLE 8
Test Accuracy of Different Variants of VGG16, and ResNet18 on CIFAR-10 With Periodic Sparse Kernels Boosted Through Insertion of Periodic FC Kernels

Model	(KVS, P)	Test acc (%)	Improvement over periodic	Parameters	Parameter reduction (%)
VGG16_PSD4_P8	(5, 8)	**92.5**	+0.87	7.57 M	48.61
VGG16_PSD4_P16	(5, 16)	92.0	+0.39	7.06 M	52.1
VGG16_PSD2_P8	(7, 8)	91.9	+1.32	4.71 M	68.1
VGG16_PSD2_P16	(7, 16)	91.3	+0.74	3.99 M	72.92
VGG16_PSD1_P8	(8, 8)	91	+3.14	3.27 M	77.78
VGG16_PSD1_P16	(10, 16)	89.8	+1.97	2.46 M	83.33
VGG16_PSD4_P4	(4, 4)	92.4	+0.77	8.59 M	41.67
VGG16_PSD2_P6	(6, 6)	92	+1.42	5.18 M	64.81
VGG16_PSD1_P9	(9, 9)	91.05	+3.22	3.09 M	79
ResNet18_PSD4_P8	(5, 8)	92.9	+0.00	5.82 M	47.83
ResNet18_PSD4_P16	(5, 16)	92.8	-0.15	5.43 M	51.26
ResNet18_PSD2_P8	(7, 8)	92.5	+1.09	3.68 M	67
ResNet18_PSD2_P16	(7, 16)	92.3	+0.81	3.15 M	71.78
ResNet18_PSD1_P8	(8, 8)	92.5	+2.84	2.62 M	76.56
ResNet18_PSD1_P16	(10, 16)	92.0	+2.4	2.01 M	82.02
ResNet18_PSD4_P4	(4, 4)	**93.0**	+0.1	6.58 M	41
ResNet18_PSD2_P6	(6, 6)	92.4	+0.9	4.04 M	63.8
ResNet18_PSD1_P9	(9, 9)	92.2	+2.6	2.48 M	77.77

TABLE 9
Test Accuracy of Different Variants of VGG16, and ResNet18 on Tiny ImageNet With Periodic Sparse Kernels Boosted With Periodic FC Kernels

Model	(KVS, P)	Top 1 acc (%)	Improvement over periodic	Parameters	Parameter reduction (%)
VGG16_PSD4_P8	(5, 8)	**57.3**	+0.35	7.57 M	48.61
VGG16_PSD4_P16	(5, 16)	56.9	+0.0	7.06 M	52.1
VGG16_PSD2_P8	(7, 8)	55.9	+1.95	4.71 M	68.1
VGG16_PSD2_P16	(7, 16)	55.5	+1.55	3.99 M	72.92
VGG16_PSD1_P8	(8, 8)	55.3	+3.55	3.27 M	77.78
VGG16_PSD1_P16	(10, 16)	55.1	+3.3	2.46 M	83.33
VGG16_PSD4_P4	(4, 4)	57.3	+0.35	8.6 M	41.67
VGG16_PSD2_P6	(6, 6)	56.3	+2.35	5.18 M	64.81
VGG16_PSD1_P9	(9, 9)	55	+3.2	3.09 M	79
ResNet18_PSD4_P8	(5, 8)	61.8	-0.09	5.82 M	47.83
ResNet18_PSD4_P16	(5, 16)	61.7	-0.23	5.43 M	51.26
ResNet18_PSD2_P8	(7, 8)	60.6	-0.13	3.68 M	67
ResNet18_PSD2_P16	(7, 16)	60.2	-0.48	3.15 M	71.78
ResNet18_PSD1_P8	(8, 8)	60.0	+1.15	2.62 M	76.56
ResNet18_PSD1_P16	(10, 16)	59.0	+0.15	2.01 M	82.02
ResNet18_PSD4_P4	(4, 4)	**62.9**	+1.0	6.58 M	41
ResNet18_PSD2_P6	(6, 6)	60.5	-0.23	4.04 M	63.8
ResNet18_PSD1_P9	(9, 9)	59.6	+0.75	2.48 M	77.77

to the non-boosting configurations used in Table 7. We tested the same sparse kernel variants as those used in Section 5.3. Thus, when the number of unique variants are less than P, we randomly chose some of the sparse kernel variants to repeat before placing the FC kernels. However, for simulation of aaa_PSD1_P8 models we randomly choose 7 of 9 unique sparse kernel variants. Note that because each period will now contain one FC kernel, the proposed criteria of covering all kernel entries within a period is automatically satisfied.

Tables 8 and 9 show the classification accuracy improvement compared to their sparse periodic counterparts and parameter count reduction compared to the corresponding baseline models. The results show that boosting yields an improvement of up to 3.2% (3.6%) in classification accuracy for CIFAR-10 (Tiny ImageNet). With sparse KSS of 4, the average performance improvement compared to periodic sparse models is $\sim$0.3%. This is quite intuitive as the potential improvement is lower when KSS is high. However, for low KSS the average improvement is $\sim$2.3%. For example, ResNet18 with KSS of 1 and repeating FC kernels with a period of 8 on CIFAR-10 provides an accuracy degradation of only $\sim$0.4% compared to the baseline, which was earlier $\sim$3.3% without the FC kernels inserted. This motivates the use of boosted pre-defined kernels that are very sparse. We observed similar trends with Tiny ImageNet as well. The relative cost of the increase in parameters due to boosting is low and, as the periodicity of the fully connected kernel placement increases, it becomes negligible. Fig. 10 shows the accuracy versus FLOPs[9] relation for different architecture variants. Models whose points lie towards the top-left have better accuracy with fewer FLOPs. In particular, for VGG16 and ResNet18 variants on CIFAR-10 and VGG16 variants on Tiny ImageNet, boosting performs consistently well, whereas, as we can see from Fig. 10d, boosting is not as beneficial for Tiny ImageNet on ResNet18. In general, we see that, with modest computation overhead, boosting consistently improves accuracy for models with extremely low KSS and maintains high accuracy otherwise.

It is important to emphasize that the overall parameter overhead is a function of both periodicity and KSS, as exemplified by the four sparse models described in Table 10 analyzed using the storage requirement formulas in Table 4. Comparing models 1 and 2, which have the same sparse KSS, shows the impact of periodicity; as does comparing models 3 and 4. In contrast, comparing models 1 and 3 shows the impact of KSS for fixed periodicity; as does comparing models 2 and 4. The last two columns of the table represent the parameter counts normalized with respect to the baseline model. Averaging across the four examples, the table shows that CSR_P reduces the overall parameter count compared to CSR, including the sparse matrix representation, by 22%. Perhaps more importantly, the results show that the CSR_P format can reduce the overall parameter count by as much as 70% compared to the baseline model.

To better evaluate the space and choice of KVs, we generated model variants with six different random seeds. We tested VGG16 and ResNet18 models with KSS of 4 and 2 to classify CIFAR-10 and Tiny ImageNet. We observed differences of less than 1% between the minimum and maximum classification accuracy across the different seeds. In particular, for ResNet18_PSD2_P8 and ResNet18_PSD4_P8 the gaps between minimum and maximum accuracy are 0.55 and 0.44%, respectively, averaged over the two datasets. For VGG16_PSD2_P8 and VGG16_PSD4_P8 these values are 0.65 and 0.65%, respectively.

Lastly, to demonstrate boosting has general benefits, Table 11 shows the results of boosting with Tiny ImageNet[10]

9. We consider FLOPs associated with only the convolution layers because they generally represent the vast majority of FLOPs.

10. For the CIFAR-10 dataset we obtained similar results, with ResNet18_pSC4_P8 exceeding the baseline performance with an average test accuracy of 92.95%.

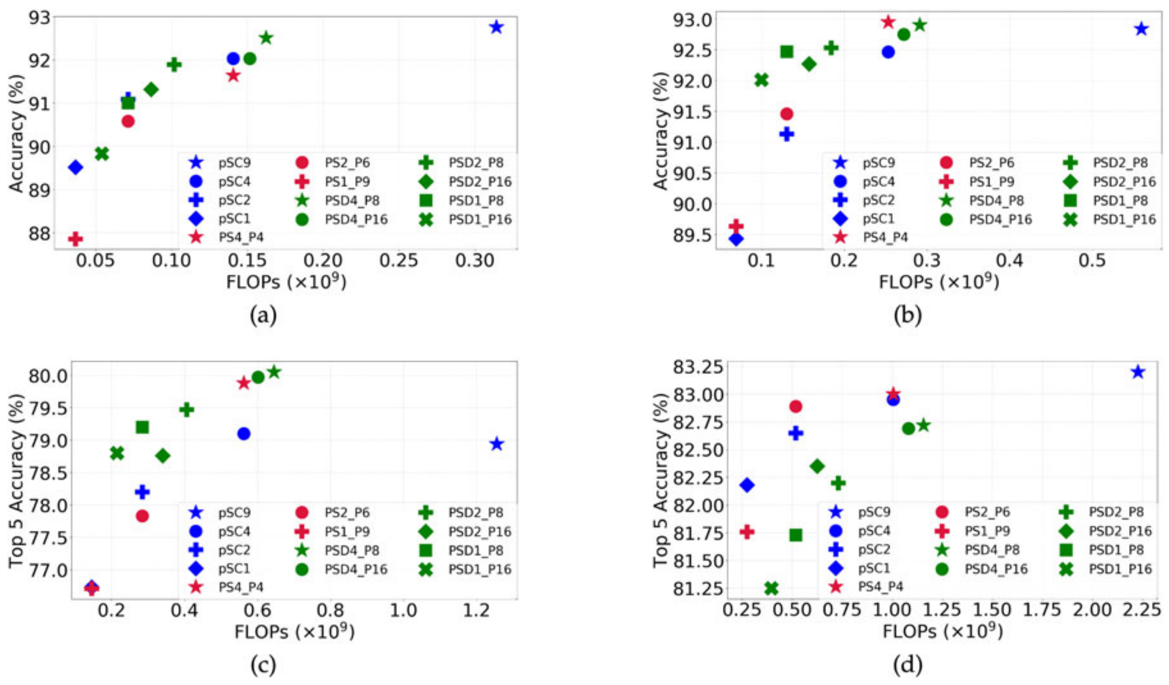


Fig. 10. Test accuracy versus FLOPs count plots for different datasets on different architectures: CIFAR-10 on (a) VGG16, (b) ResNet18 variants; Tiny ImageNet on (c) VGG16, (d) ResNet18 variants.

when the FC kernels are placed periodically, with period P_D, in between sparse kernels *with no pre-defined KVS or kernel variants* (as described in Section 5.2). Note, as with the networks described in Section 5.2, the lack of structure makes these models have higher indexing overhead compared to the periodic models analyzed above.

TABLE 10
Parameters Reduction and Corresponding Normalized Storage Requirement Including Indexing Overhead for Four VGG16 Variants With Both CSR_P and CSR Format of Compressed Storage

No.	Model	Model param. reduction (%)	Normalized param. count, using CSR_P	Normalized param. count, using CSR
1	VGG16_PSD4_P8	48.61	0.66	0.85
2	VGG16_PSD4_P16	52.10	0.69	0.81
3	VGG16_PSD1_P8	77.78	0.34	0.42
4	VGG16_PSD1_P16	83.33	0.30	0.35

TABLE 11
Test Accuracyn of Boosting as a General Method to Improve Accuracy

Model	(KVS, P_D)	Top 1 acc (%)	Parameters	Parameter reduction (%)
VGG16_pSC4_P8	(–, 8)	**56.6**	7.57 M	48.61
VGG16_pSC4_P16	(–, 16)	56.2	7.06 M	52.1
VGG16_pSC2_P8	(–, 8)	56.6	4.71 M	68.1
VGG16_pSC2_P16	(–, 16)	56.4	3.99 M	72.92
VGG16_pSC1_P8	(–, 8)	55.5	3.27 M	77.78
VGG16_pSC1_P16	(–, 16)	54.8	2.46 M	83.33
ResNet18_pSC4_P8	(–, 8)	61.8	5.82 M	47.83
ResNet18_pSC4_P16	(–, 16)	**62.3**	5.43 M	51.26
ResNet18_pSC2_P8	(–, 8)	61.3	3.68 M	67
ResNet18_pSC2_P16	(–, 16)	60.5	3.15 M	71.78
ResNet18_pSC1_P8	(–, 8)	59.8	2.62 M	76.56
ResNet18_pSC1_P16	(–, 16)	59.2	2.01 M	82.02

Dataset used here is Tiny ImageNet.

5.5 Performance Comparison With ShuffleNet and MobileNetV2

Because ShuffleNet [31] and MobileNetV2 [30] are two widely-accepted low-complexity CNN architectures, we compared them with our proposed pre-defined periodic sparse models that have similar or fewer FLOPs.[11] In particular, Fig. 11a shows that for CIFAR-10 the ResNet18_PSD1_P16 increases accuracy to 92% compared to the baseline MobileNetV2 (ShuffleNet) accuracy of 90.3% (~89%). Note that our obtained accuracies are also superior than reported in [47] and only around 1% less than the accuracy reported in [48] which was trained for 180 additional epochs. The pre-defined sparse CNN model VGG16_PSD1_P8 With 0.073 G FLOPs, has approximately $1.24\times$ ($1.34\times$) fewer computation complexity yet still outperforms MobileNetV2 (ShuffleNet) in terms of accuracy. For Tiny ImageNet, as shown in Fig. 11b, our best classifying model provides an accuracy improvement of 3.2% with only 4% (2.6%) increased complexity compared to MobileNetV2 (ShuffleNet).

Moreover, as we can see from Figs. 12a, and 12b, with $2.42\times$ ($1.08\times$) fewer parameters our proposed models perform similar to ShuffleNet for Tiny ImageNet (CIFAR-10). Similarly, the parameter requirement of our proposed models with similar accuracy as MobileNetV2 are $1.15\times$, and $2.38\times$ lower for CIFAR-10 and Tiny ImageNet, respectively.[12]

11. Note that we kept the hyperparameters for MobileNetV2 training the same as ResNet18 except the weight decay which was set to 0 as recommended by the original papers [30].

12. These values can be translated to the normalized parameter count with the help of the formulas in Table 4.

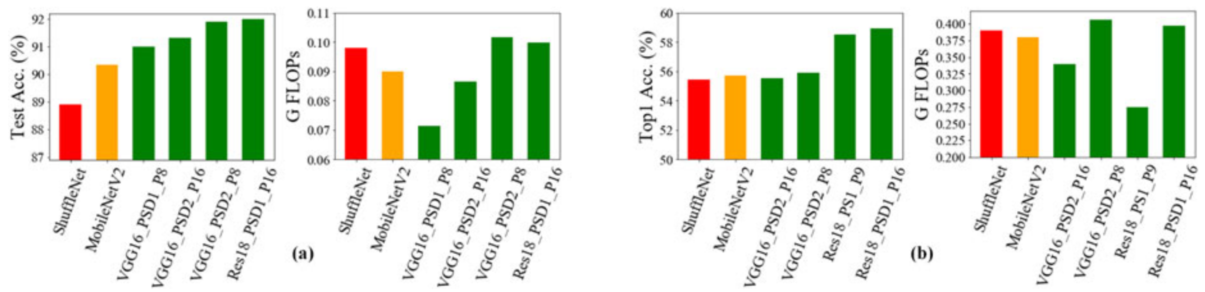


Fig. 11. Performance comparison of our proposed architectures that have similar or fewer FLOPs than ShuffleNet and MobileNetV2 with comparable or better classification accuracy on (a) CIFAR-10 and (b) Tiny ImageNet.

5.6 Performance Evaluation on Networks Models With Scaled Down Width

Squeezing the network layers, i.e., reducing the number of channels per 3D filter by a factor of α (<1.0), popularly known as the width multiplier, is another simple technique to reduce the network's FLOPs and storage requirement [29], [41], [49]. To further establish the idea of the pre-defined periodic sparsity, we apply our proposed kernels in squeezed variant of the ResNet18 architecture with an α of 0.5. The important network model parameters of the squeezed variants of ResNet18 and MobileNetV2 models are described in Table 12. With the same hyperparameter settings as stated in Section 5.1, the baseline accuracy for ResNet18 with $\alpha = 0.5$ are 91.1, and 59.1% for CIFAR-10, and Tiny ImageNet, respectively. We trained several variants of this squeezed model with KSS values of 4, 2, and 1, each with the fully connected kernel repeating after every 8 and 16 kernels. Fig. 13 shows our proposed variants of squeezed ResNet18 consistently outperforms both MobileNetV2_0.75 and MobileNetV2 in classification accuracy, keeping the number of FLOPs similar or lower. In particular, Fig. 13a shows that on CIFAR-10 dataset, to provide similar accuracy the squeezed ResNet18 with KSS of 2 and periodicity of 16 requires 2.36$\times$ fewer FLOPs compared to MobileNetV2. Also, the ResNet18 variant that requires the least number of FLOPs, provides $\sim$1% improved accuracy with 2.6$\times$ fewer computations compared to MobileNetV2_0.75. A similar trend is observed for Tiny ImageNet, as shown in Fig. 13b. Averaged over the two datasets, the proposed squeezed ResNet18 variants provides similar accuracy with 2.42$\times$, and 2.37$\times$ fewer FLOPs compared to MobileNetV2_0.75 and MobileNetV2, respectively. On the same datasets, when we constrain the number FLOPs to be similar, pre-defined periodic sparsity can provide an average accuracy improvement of $\sim$3.16% and $\sim$2.48%, compared to MobileNetV2 with α of 0.75 and 1.0, respectively. The model parameter reduction factors are proportional to the computation reduction and as the ResNet18_0.5 model has comparable parameters as MobileNetV2, advantage in storage for the sparse versions of ResNet18_0.5 is quite clear, and thus not discussed in details for brevity's sake.

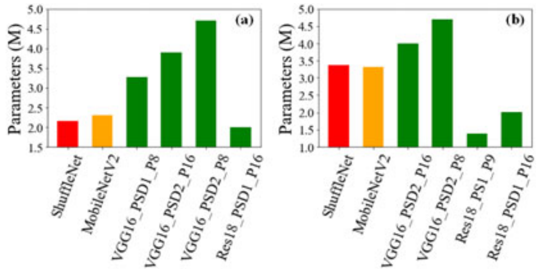


Fig. 12. Comparison of the number of model parameters of the network models described in Fig 11 for (a) CIFAR-10 and (b) Tiny ImageNet datasets.

6 Conclusion

This paper showed that with pre-defined sparsity in convolutional kernels the network models can achieve significant model parameter reduction during both training and inference without significant accuracy drops. However, managing sparsity requires matrix indexing overhead in terms of storage and energy efficiency. To address this shortcoming, we added periodicity to the sparsity, periodically using same sparse kernel patterns in the convolutional layers, significantly reduce the indexing overhead.

Furthermore, to deal with the performance degradation due to pre-defined sparsity, we introduced a low-cost network architecture modification technique in which FC kernels are periodically inserted in between sparse kernels. Experimental results showed that, compared to the sparse-periodic variants, this boosting technique improves average classification accuracy by up to $\sim$2.3%, averaged over two periodicity of 8, and 16 in ResNet18 and VGG16 architecture on CIFAR-10 and Tiny ImageNet. We also demonstrated the merits of the proposed architectures with squeezed variants of ResNet18 (width multiplier <1.0) and have shown it to outperform MobileNetV2 by an average accuracy of $\sim$2.8% with similar FLOPs.

Our future work includes exploring additional forms of compressed sparse representations and their hardware support. Lastly, we note that much of our findings are empirical in nature. Finding a more theoretical basis that can motivate and guide the use of periodic pre-defined sparsity in deep learning is also an important area of future work.

TABLE 12
CONV Layer Channel Width Parameters With Different α Values of the Network Models

Name	α	Convolution layer different channel sizes
ResNet18	1.0	[64, 128, 256, 512]
ResNet18_0.5	0.5	[32, 64, 128, 256]
MobileNetV2	1.0	[16, 24, 32, 64, 96, 160, 320]
MobileNetV2_0.75	0.75	[12, 18, 24, 48, 72, 120, 240]

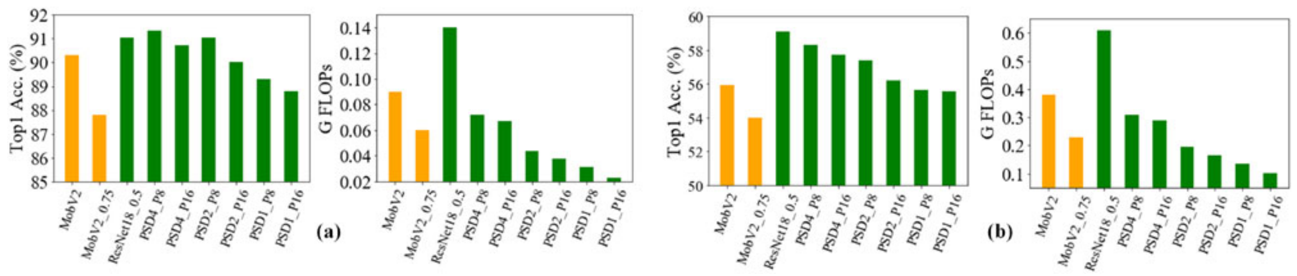


Fig. 13. Performance comparison in terms of test accuracy and FLOPs of different squeezed (width multiplier 0.5) ResNet18 variant models with MobileNetV2 (MobV2) having width multiplier 1.0 and 0.75 on (a) CIFAR-10, and (b) Tiny ImageNet.

Acknowledgments

This work was partly supported by NSF Grant #1763747 and AFRL Award #FA8750-16-2-0204. Portions of this article were presented at the 57th Allerton Conference, 2019 [1].

References

[1] S. Kundu, S. Prakash, H. Akrami, P. A. Beerel, and K. M. Chugg, "pSConv: A pre-defined sparse kernel based convolution for deep CNNs," in *Proc. 57th Annu. Allerton Conf. Commun. Control Comput.*, 2019, pp. 100–107.

[2] A. Krizhevsky *et al.*, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.

[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," 2013, *arXiv:1312.6229*.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7263–7271.

[9] H. Tao, W. Li, X. Qin, and D. Jia, "Image semantic segmentation based on convolutional neural network and conditional random field," in *Proc. 10th Int. Conf. Adv. Comput. Intell.*, 2018, pp. 568–572.

[10] A. Coates *et al.*, "Deep learning with COTS HPC systems," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[12] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture*, 2016, pp. 243–254.

[13] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

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

[15] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, June 2017.

[16] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5687–5695.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[18] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[19] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv: 1810.05270*.

[20] T. Zhang *et al.*, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 184–199.

[21] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv: 1702.03044*.

[22] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3466–3473.

[23] A. Ren *et al.*, "ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2019, pp. 925–938.

[24] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2547–2554.

[25] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

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

[27] S. Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg, "Pre-defined sparse neural networks with hardware acceleration," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 332–345, Jun. 2019.

[28] A. Fayyazi, S. Kundu, S. Nazarian, P. A. Beerel, and M. Pedram, "CSrram: Area-efficient low-power ex-situ training framework for memristive neuromorphic circuits based on clustered sparsity," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2019, pp. 465–470.

[29] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

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

[32] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, Canada, Tech. Rep. TR-2009, 2009.

[33] L. Hansen, "Tiny ImageNet challenge submission," *CS 231N*, 2015.

[34] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 38, pp. 247–257, 2010.

[35] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[36] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, Contour and Grouping in Computer Vision*, Berlin, Germany: Springer, 1999, pp. 319–345.

[37] V. Vanhoucke, "Learning visual representations at scale," *ICLR Invited Talk*, vol. 1, 2014, Art. no. 2.

[38] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep Roots: Improving CNN efficiency with hierarchical filter groups," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1231–1240.

[39] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4278–4284.

[40] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1492–1500.

[41] 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 <0.5MB model size," 2016, *arXiv:1602.07360*.

[42] J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming*. Hoboken, NJ, USA: Wiley, 2014.

[43] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–12.

[44] X. Ma *et al.*, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," 2019, *arXiv:1909.05073*.

[45] M. Greenberg, "LPDDR3 and LPDDR4: How low-power DRAM can be used in high-bandwidth applications," by JEDEC, 2013.

[46] A. Paszke *et al.*, "Automatic differentiation in pytorch," 2017.

[47] T. Lawrence and L. Zhang, "IoTNet: An efficient and accurate convolutional neural network for IoT devices," *Sensors*, vol. 19, no. 24, 2019, Art. no. 5541.

[48] X. She, Y. Long, D. Kim, and S. Mukhopadhyay, "ScieNet: Deep learning with spike-assisted contextual information extraction," 2019, *arXiv: 1909.05314*.

[49] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2019, *arXiv: 1905.11946*.

Souvik Kundu (Member, IEEE) received the BTech degree in electronics and communication engineering from the West Bengal University of Technology, Kolkata, West Bengal, India, in 2009 and the MTech degree in microelectronics and VLSI design from Indian Institute of Technology Kharagpur, India, in 2015. He is currently working towards the PhD degree in electrical and computer engineering at the University of Southern California, Los Angeles, California. He worked as an R&D engineer II at Synopsys India Pvt. Ltd., and as a digital design engineer at Texas Instruments India Pvt. Ltd., from 2015 to 2016 and from 2016 to 2017, respectively. His research interests include energy aware sparsity, model search, and algorithm-hardware co-design of neural networks in machine learning.

Mahdi Nazemi (Member, IEEE) received the BS degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2014. He is currently working toward the PhD degree in electrical engineering at the University of Southern California, Los Angeles, California. He was an R&D software engineering intern at the Cadence's Genus Synthesis Solution group during the summer of 2016 and a software engineering intern at the Microsoft's Artificial Intelligence and Research engineering group during the summer of 2018. His research interests include algorithm/hardware co-design for the efficient processing of machine learning algorithms including deep neural networks. One of his recent publications in this area won the Best Paper Award at the Asia and South Pacific Design Automation Conference, in 2019.

Massoud Pedram (Fellow, IEEE) received the BS degree in electrical engineering from Caltech, Pasadena, California, in 1986 and the PhD degree in electrical engineering and computer sciences from the University of California, Berkeley, Berkeley, California, in 1991. Subsequently, he joined the Department of Electrical Engineering at the University of Southern California, Los Angeles, California where he currently holds the Charles Lee Powell chair. His research interests include computer-aided design of VLSI circuits and systems, low power electronics, energy-efficient processing, electrical energy storage systems, power conversion and management ICs, quantum computing, and superconductive electronics. He has authored four books and more than 700 archival and conference papers. He received the 2015 IEEE Circuits and Systems Society Charles A. Desoer Technical Achievement Award for his contributions to modeling and design of low power VLSI circuits and systems, and energy efficient computing and the 2017 USC Viterbi School of Engineering Senior Research Award. He also received the Third Most Cited Author Award at the 50th anniversary of the Design Automation Conference June 2013.

Keith M. Chugg (Fellow, IEEE) received the BS degree (high distinction) in engineering from Harvey Mudd College, Claremont, California, in 1989 and the PhD degree in electrical engineering from the University of Southern California (USC), Los Angeles, California, in 1995. Since 1996, he has been on the faculty of the Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, California, where he is currently a professor. His research interests are in the general areas of signal processing, digital communications, machine learning, and associated efficient implementations. He is a co-founder of TrellisWare Technologies, Inc., where he serves as a chief scientist.

Peter A. Beerel (Senior Member, IEEE) received the BSE degree in electrical engineering from Princeton University, Princeton, New Jersey, in 1989 and the MS and PhD degrees in electrical engineering from Stanford University, Stanford, California, in 1991 and 1994, respectively. He then joined the Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, California where he is currently a professor and the associate chair of the Computer Engineering Division. He is also a research director at the Information Science Institute at USC. Previously, he cofounded TimeLess Design Automation to commercialize an asynchronous ASIC flow, in 2008 and sold the company, in 2010 to Fulcrum Microsystems which was bought by Intel in 2011. His research interests include a variety of topics in computer-aided design, machine learning, hardware security, and asynchronous VLSI and the commercialization of these technologies.