IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EDGE COMPUTING AND NETWORKING FOR UBIQUITOUS AI

Received April 1, 2020, accepted April 23, 2020, date of publication May 8, 2020, date of current version May 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993506

IKW: Inter-Kernel Weights for Power Efficient Edge Computing

PRAMOD UDUPA[1], (Senior Member, IEEE), GOPINATH MAHALE[1], KIRAN KOLAR CHANDRASEKHARAN[1], AND SEHWAN LEE[2]

[1]Samsung Advanced Institute of Technology (SAIT), Samsung Research and Development Institute India-Bangalore Pvt., Ltd., Bengaluru 560037, India
[2]Samsung Advanced Institute of Technology (SAIT), Samsung Electronics Company, Ltd., Suwon 16678, South Korea

Corresponding author: Pramod Udupa (pramod.udupa@samsung.com)

ABSTRACT Deep Convolutional Neural Networks (CNN) have achieved state-of-the-art recognition accuracy in a wide range of computer vision applications like image classification, object detection, semantic segmentation etc. Applications based on CNN require millions of multiply-accumulate (MAC) operations to be performed between input pixels and kernel weights during inference. This work investigates a technique, which can be used to eliminate redundant multiplications for a subset of kernel weights in a CNN layer by utilizing identical and/or similar inter-kernel weights (IKW) across kernels. In this work, IKW technique is used to identify identical and/or similar inter-kernel weights in trained, unpruned/pruned, quantized CNN kernels before inference phase. After identification of identical and/or similar inter-kernel weights, a subset of kernel weights termed non-pivot kernel weights are made zero, the other subset called pivot kernel weights are left unchanged. The multiplication corresponding to non-pivot kernel weights are eliminated, thus reducing computations. The products corresponding to non-pivot kernel weights are supplied by multiplication operation of pivot kernel weights, and hence causing no degradation in inference accuracy. Through experiments on state-of-the-art CNNs, we demonstrate that application of IKW technique enhances kernel sparsity by 9-37% for 8-bit precision kernel weight and 18-43% for 4-bit precision kernel weight without degrading the recognition accuracy of the CNN model. Enhanced kernel sparsity can be used to save power by clock gating the compute unit, or increase execution performance by skipping computations pertaining to zero valued non-pivot kernel weights. In addition, power savings are achieved by eliminating redundant power expensive fixed-point multiplication operations. The practical utility of the IKW technique is demonstrated by mapping it to well-known state-of-the-art CNN accelerator architectures. Mapping of the IKW technique on existing CNN accelerator architectures shows reduction in power by at least 12% for 8-bit precision and 19% for 4-bit precision kernel weight. Improvement in execution performance by at least 2% for 8-bit precision and 13% for 4-bit precision kernel weight is observed.

INDEX TERMS Inter-kernel weights, quantization, multiply-accumulate unit, split accumulator, kernel zero skipping, convolutional neural network, kernel pruning, identical kernel weights, similar kernel weights.

I. INTRODUCTION

Artificial Intelligence (AI) applications based on Deep Neural Networks (DNNs) have become pervasive due to their near human level performance in diverse application domains [1], [2] such as image classification, object detection [3], [4], scene understanding [5] etc. DNNs have enjoyed renaissance [6] after many years due to possible factors like increased compute capacity for training DNNs which can process large amounts of data (e.g. graphic processing units) and availability of huge amount of digital data [7], [8] for training DNNs. The intersection of above two factors enabled DNNs to be trained in an end-to-end manner which made the DNN features learnt more robust compared to hand designed kernels to extract features from the data.

DNNs use Convolutional Neural Networks (CNN) as the major workhorse for extracting useful information from image and video, while Recurrent Neural Networks (RNN) are heavily used for working with text and voice based inputs. CNNs have been able to improve their classification

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaofei Wang.

performance by stacking large number of layers [9], [10] and process larger inputs, to better extract information from a bigger dataset. With CNNs getting bigger, requirement of multiply-accumulate (MAC) operations needed to be performed for real-time inference has crossed millions of operations per second. To be able to perform these compute intensive operations on battery constrained devices like mobile phones is one of the motivating factors for power efficient computation [11] of CNNs.

Broadly speaking, the main motivation factor is to enable power efficient edge computing [12], which means improving the power efficiency for all computing devices on the edge of the network. Devices on the edge of network, have limited compute and battery power, but produce/collect a large amount of data, which need processing to extract information. In edge computing, a lot of energy is spent to transmit data to cloud for processing. Further, it waits for the results to come back to take some action, which increases latency for the application on the edge. Privacy is also a concern when data is being sent to cloud. Since devices at the edge produce a lot of data, which require real-time inference for enabling AI applications, supporting power efficient CNN computations becomes a priority. Considering these motivation factors, we have concentrated our efforts in finding generalized techniques for improving power efficiency of CNN computations on edge devices for inference operation.

To meet the requirement of power efficient computations on edge devices, several algorithmic optimizations and specialized CNN accelerator hardware architectures have been proposed for CNN inference [13]–[19]. Algorithmic optimizations for CNN have majorly concentrated on pruning of kernel weights [20]–[25] to reduce the number of non-zero kernel weights without significant degradation in accuracy. Pruning reduces the model size which helps in reducing storage size, thus saving energy for transferring kernel weights from memory to compute unit. Pruning increases kernel sparsity which can be used by the hardware for skipping ineffectual computations for improving performance and power consumption. But, pruning requires re-training of the CNN which is computationally demanding and infeasible on edge devices. Moreover, pruning beyond a particular point results in significant degradation of accuracy.

Quantization of kernel weights to fixed-point precision has been actively pursued to reduce the cost of hardware computations during inference. Quantization is complementary to pruning and generally applied after pruning, to trained CNN models. In this stage, the trained kernel weights in floating point numbers are quantized [26] and approximated to low precision fixed point numbers. Fixed point operations result in cheaper hardware with respect to power [27] and area, compared to floating point operations. Lin *et al.* [28] and Judd *et al.* [29], showed that quantizing the trained kernel weights and IFMs to 8 bits did not result in a major fall in top-1 recognition accuracy in state-of- the-art CNNs during inference. Similar results for 4-bit quantization are demonstrated [30], [31] in literature. To further improve the energy efficiency,

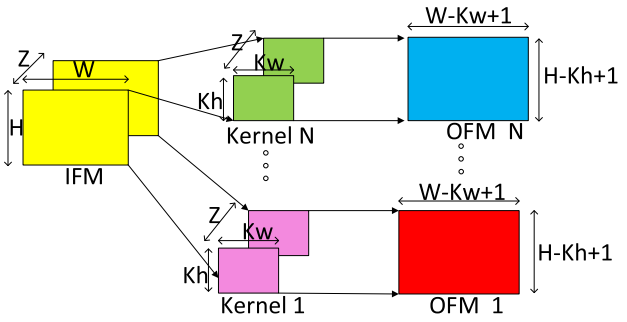


FIGURE 1. (a) Illustration of 3-D Convolution operation in CNN layer. W, H: width,height of IFM, Z: number of IFM/kernel channels, Kw, Kh: width,height of kernel, N: number of kernels.

there are methods in literature such as quantizing data using limited number of weights [32], which is hard to generalize to different CNNs.

In all the CNN hardware accelerator architectures proposed in literature, the CNN accelerator hardware has been optimized for the 3-D Convolution operation which takes up nearly 90% of the computations [33], [34] in the CNN. 3-D Convolution operation has a very regular computation structure where 3-D Input Feature Map (IFM) is convolved with 3-D kernel to produce an Output Feature Map (OFM) channel. DianNao [13] and DaDianNao [14] were the initial works which utilized the parallel computation structure present in convolution for acceleration. Eyeriss [16] adopted a row-stationary approach to improve IFM re-use to reduce the required memory bandwidth and memory access power. NVDLA [18] was the first architecture to utilize channel-first storage format to store IFM, kernel and OFM. NVDLA could reduce the power consumption by switching off the computation circuitry in case of zero valued IFM or kernel. Cnvlutin [17] further enhanced the value based usage by using zero valued IFM to skip ineffectual multiplications which improved execution performance. This approach is different compared to architectures like NVDLA, DaDianNao which do not use zero valued IFM or kernel to improve execution performance. ZeNA [19], [35] architecture uses both zero valued IFM and kernel to further improve performance and reduce power consumption. We see a trend in newer accelerators in using value based approach to eliminate ineffectual computations to improve performance and reduce power dissipation.

Considering the compute limitations on devices at the edge, alternate approaches have been proposed to increase sparsity of CNN model by utilizing the computation structure of convolution operation. These alternate approaches build on top of pruning to further enhance CNN sparsity without degrading the classification accuracy. These methods have resulted in power savings and improved performance. A good example for this approach is UCNN [36], which targets to avoid redundant multiplications of identical kernel weight within a kernel by factorizing the dot product operation. UCNN proposes an architecture which stores the dot product value produced by a repeated kernel weight with an IFM

pixel, in memory. This dot product is read back when repeated kernel weight needs to be multiplied with the same IFM pixel. To realize this, repeated weights in CNN kernels are searched and indexed, which are then utilized during inference process for avoiding repeated redundant multiplications. The work in UCNN explored identical kernel weights within a kernel exposing a limited scope for finding redundant multiplications. In this work, we investigate possibility of using identical and similar kernel weights across kernels, termed inter-kernel weights to increase the sparsity of kernels for improved power and performance, which to the best of our knowledge has not been explored before.

In this work, we propose a scheme to detect Inter-Kernel Weights (IKW) that enhances the kernel sparsity for convolution and Fully Connected (FC) layers of CNNs. We also propose hardware architecture for IKW, which uses the metadata generated during IKW search process to improve execution performance and reduce power consumption. The proposed IKW Architecture can be integrated in all state-of-the-art CNN accelerators with or without kernel zero skipping feature. The main contributions of our work are as follows:

- We propose a search procedure for detecting Identical Inter-Kernel Weights (IIKW) and Similar Inter-Kernel Weights (SIKW) in CNNs for enhancing kernel sparsity. We describe metadata format for encoding presence of IKW in kernels.
- We propose *IKW Architecture*, which is a hardware architecture supporting the usage of IKW to eliminate redundant multiplications in MAC operations by using the metadata generated during the IKW search procedure. We propose two templates of IKW Architecture depending on whether the underlying CNN architecture on which IKW Architecture is applied, uses kernel zero skipping or not.
- We propose split accumulator optimization to reduce the power consumption in accumulator of MAC operation
- We detail the introduction of the proposed IKW Architecture templates in state-of-the-art CNN accelerators (NVDLA, Cnvlutin, ZeNA). We measure power, area and performance for the state-of-the-art CNN accelerators with and without IKW Architecture on different CNNs to quantify the impact of IKW.

The rest of the paper is organized as follows. Section II introduces the concept of identical and similar Inter-Kernel Weights (IKW), as forms of IKW, in the context of convolution and FC layers. The section details the IKW search procedure and generation of metadata for detected IKW in state-of-the-art CNNs. We discuss in detail results of detected IKW in unpruned and pruned kernels at 4-bit and 8-bit precision for standard CNNs like VGG-16 [37], Inception-v3 [1], Inception-v4 [38]. Section III illustrates the proposed *IKW Architecture* templates for utilizing detected IKW to eliminate redundant multiplications in MAC. Sections III-C, III-D and III-E explain the introduction of the proposed *IKW Architecture* in state-of-the-art CNN accelerators like NVDLA, Cnvlutin and ZeNA respectively. We discuss the impact of introduction of the proposed *IKW Architecture* on power, performance and area for standard CNNs like VGG-16 and Inception-v4. Section IV concludes the paper.

II. INTER-KERNEL WEIGHTS (IKW)

A. PRELIMINARIES

A typical CNN consists of a series of linear computation operation layers separated by non-linear operators in between, which can be represented by a directed acyclic graph [39]. Each linear computation layer consists of multiple kernels (filters) working on the same input IFM to extract different features from it. Linear operator layers in the beginning of the CNN have kernels which extract simpler features [40] like edge, horizontal line, circle etc., while kernels at the later layers look for more complicated shapes [40] like human nose, face, car etc. CNNs extract information from a given image in a hierarchical manner. Deeper CNNs [9] are able to extract more information compared to shallow CNNs. Figure 1 illustrates a typical 3-D Convolution operation in CNNs, where single 3-D IFM is convolved with N different 3-D kernels to produce N channels of OFM. OFM produced by set of N kernels together is also 3-D, as shown in Figure 1.

As CNNs have evolved, the number of layers in a CNN has increased from 5 [41] to hundreds of layers [10] in newer CNNs. Along with the increase in number of layers in CNN, the number of kernels per layer has also increased from AlexNet [41] to latest CNNs to extract more information in each layer to improve recognition accuracy. Due to these factors, the computational complexity of CNNs has increased significantly. One possible way to reduce the computational complexity is to eliminate redundant computations in convolution operation without affecting the recognition accuracy. Based on the observation that single 3-D IFM is convolved with N different kernels in every layer, we see an opportunity to eliminate redundant multiplication of IFM pixel with kernel weight whose value may be repeated across multiple kernels. The definition of kernel weight which multiplies with the same IFM pixel across two kernels, termed as Inter-Kernel Weight (IKW), is formalized in Section II-B

B. IKW CONCEPT

The concept of IKW is based on the observation that kernel weights having the same co-ordinate and same channel number across two kernels in a CNN layer multiply with same IFM pixel. The observation is applicable to convolution and FC layers used in CNN/RNN. For example, consider convolution with three kernels k_0, k_1, k_2 of dimension $3 \times 3 \times 1$ as shown in Figure 2(a) belonging to a single CNN layer. As per the convolution operation, kernel weight at location $(0, 0, 0)$ of kernels k_0, k_1, k_2 multiply with the same IFM pixel. If the kernel weights are identical, as in location $(1, 0, 0)$ of k_0 and k_1, then the product of multiplication will also be same. Due to kernel weights being identical, products generated by weight in location $(1, 0, 0)$ of k_0 can be shared with

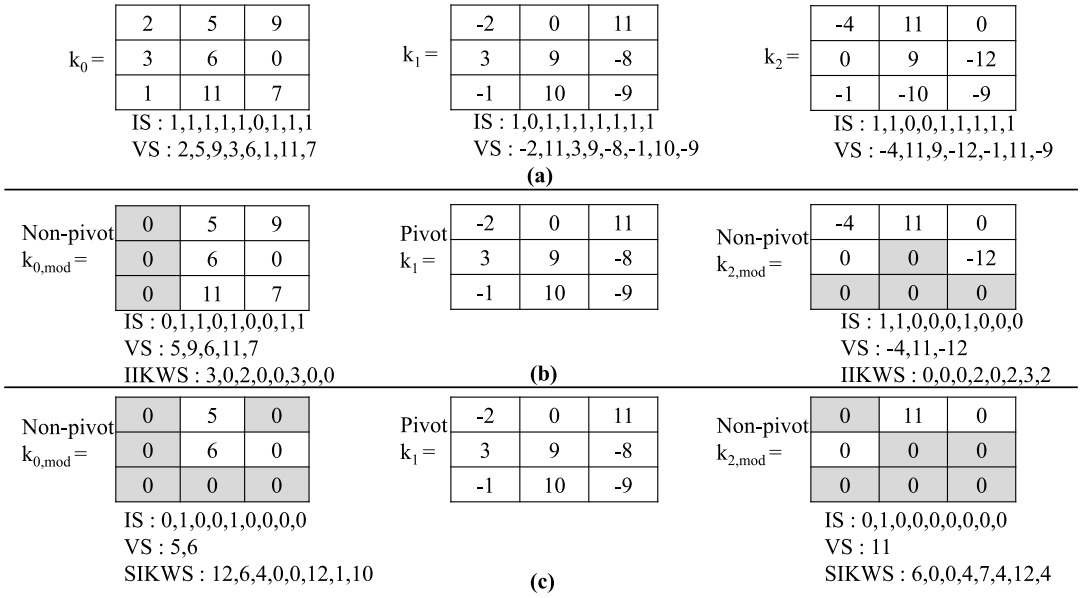


FIGURE 2. (a) Kernels before IKW search procedure. IS: Index Stream, VS: Value Stream, IIKWS: Identical IKW stream, SIKWS: Similar IKW stream. (b) Kernels after IIKW search procedure showing modified non-pivot kernels and unmodified pivot kernel. Grey shaded cells in modified non-pivot kernels represent zeros introduced due to IIKW search. IIKW stream is introduced in non-pivot kernels and is encoded with respect to pivot kernel. (c) Kernels after SIKW search procedure showing modified non-pivot kernels and unmodified pivot kernel. Grey shaded cells in non-pivot kernels represent zeros introduced due to SIKW search. SIKW stream is introduced in non-pivot kernels and is encoded with respect to pivot kernel. Note that number of entries in IIKWS or SIKWS is equal to number of non-zero entries in pivot kernel.

k_1 or vice-versa. This helps in reduction of computations, by avoiding redundant multiplication operations, without changing the values of the OFM pixels. The reduction in multiplication is applicable to computation of every OFM pixel generated by the kernel, which is reusing the product generated by a different kernel. Hence, identical inter-kernel weight is able to eliminate multiplications of MAC operation. We term two non-zero weights in two different kernels as IKW if and only if

- they have the same channel number
- they have the same planar co-ordinates

IKW is used during CNN inference to minimize number of multiplications. CNN inference is typically done with lower bit-width, e.g., 8 bit precision for kernel weight. Since the number of unique weights possible with 8 bit precision is 256, and the number of kernel weights in a CNN layer is typically greater than 256 [36], repeated kernel weights are highly possible across kernels in all the CNN layers. With the trend of CNN inference moving towards lower precision than 8 bits, the probability of repeated kernel weights increased significantly. In section II-C, we detail the experimental setup for detecting inter-kernel weights in state-of-the-art CNNs. Section II-D explains the search procedure of IIKW in state-of-the-art CNNs. Section II-E extends the search procedure to SIKW.

C. IKW EXPERIMENTAL SETUP

For evaluating the idea of IKW in state-of-the-art CNNs, we have used Caffe [42] tool for extracting the information about each layer of the CNN and to evaluate the classification accuracy of the CNN model. Trained, unpruned Caffe models of the CNNs were taken from Caffe Model Zoo [42]. We selected VGG-16 [37], Inception-v3 [1], Inception-v4 [38] for IKW search. Classification accuracy for each of the selected CNNs was measured by using 50000 images from the test set of the ImageNet database using the Caffe framework. Classification accuracy of the unpruned CNN models on ImageNet database working in floating-point precision were computed in Caffe, as shown in Table 1.

Next, unpruned CNN models were quantized to 8-bit precision by using Samsung Python libraries for quantization. During quantization, every channel of the kernel was quantized independently based on the maximum value in that channel. Maximum value was used to select the number of integer bits required, while remaining bits were used for fractional part. Further, unpruned CNN models were quantized to 4-bit precision except the first and last layers, which were kept at floating-point precision. Since direct quantization to 4-bit results in significant dip in classification accuracy, the 4-bit quantized CNN model was re-trained to reach classification accuracy close to 8-bit unpruned CNN models. Classification accuracy of both 8-bit and 4-bit quantized CNN models were measured in Caffe, as shown in Table 1.

To measure IKW in pruned kernels, the unpruned CNN models were pruned using Samsung Python libraries for pruning. Pruning was carried out using threshold based method. This method prunes kernel weights to zero below a particular threshold. Pruning is an iterative process, where determination of threshold value is dependent on the target classification accuracy. Classification accuracy of the pruned

TABLE 1. Top-1 Accuracy of CNNs used for IKW on ImageNet database.

CNN	Floating Point	Fixed Point (8-bit)	Fixed Point (4-bit)
Unpruned CNNs			
VGG-16 [37]	68.45	68.5	62.4
Inception-v3 [1]	78.55	78.25	75.2
Inception-v4 [38]	79.6	79.13	76.1
Pruned CNNs			
VGG-16 [37]	68.5	68.3	62.2
Inception-v3 [1]	78.1	77.92	74.89
Inception-v4 [38]	79.35	79.39	75.7

TABLE 2. Variants of quantized CNN models used for IKW search.

Model Sparsity	Precision	
Unpruned	8-bit	4-bit
Pruned	8-bit	4-bit

model is shown in Table 1. It can be observed that classification accuracy of the pruned models is within 0.5% of the accuracy of floating point unpruned models. Similar to quantization of unpruned CNN models, the pruned CNN models were quantized to 8-bit and 4-bit precision. Classification accuracy for quantized, pruned CNN models were noted, as given in Table 1. We now have four variations of quantized CNN models for IKW search, as shown in Table 2. These four variations of CNN model were passed through IKW search module. IKW search module was coded in Python and works on quantized layers of the CNN model. It should be noted that IKW search procedure and metadata generation is done offline after kernel quantization step before inference. The generated metadata can be used during CNN inference any number of times. Note that IKW does not degrade classification accuracy of the underlying model, since OFM values produced are exactly same as that produced without using IKW. Section II-D details the IKW search procedure on CNN models searching for IIKW.

D. IDENTICAL INTER-KERNEL WEIGHTS (IIKW)

We define Identical IKW as IKW whose magnitudes are identical with same or opposite signs. Consider the example of three kernels k_0, k_1, k_2 of dimensions $3 \times 3 \times 1$ as shown in Figure 2(a). Compressed representation of each kernel is given below the kernel matrices. The compressed representation consists of index stream (IS) and value stream (VS). IS indicates whether the kernel location contains a zero valued kernel weight (indicated by entry 0) or non-zero valued kernel weight (indicated by entry 1). VS stores only non-zero valued kernel weights. As can be observed from Figure 2(a), the length of IS array is same as the kernel dimension, while length of VS array is variable, depending on the number of non-zero values in the kernel. Given three kernels of Figure 2(a), IIKW search procedure is as follows:

- Compare IKW of kernels k_0 and k_1 with a condition that magnitude is identical independent of sign. Note down the number of IIKW and update the score for k_0, k_1. Score indicates the number of IIKW found during the comparison. After first comparison, score for k_0, k_1 is 3, 3 respectively.
- Compare IKW of kernels k_0 and k_2 and update score of k_0, k_2. Score for k_0 is 4, k_2 is 1 after second comparison
- Compare IKW of kernels k_1 and k_2 and update score of k_1, k_2. Score for k_1 is 7, k_2 is 4 after third comparison

TABLE 3. IIKW encoding for IIKW stream of non-pivot kernel weight (y) with respect to Pivot kernel weight (x).

IKW Encoding	Description	Comment
0	$y \neq x$	Not an IIKW
2	$y = x$	Same magnitude, same sign
3	$y = -x$	Same magnitude, opposite sign
1	Not Applicable	Demarcation

After all comparisons, IIKW search score of k_0, k_1, k_2 is 4,7,4 respectively. The kernel with highest score is selected as the *pivot kernel*, while remaining kernels are designated *non-pivot* kernels, as shown in Figure 2(b). Pivot kernel k_1 entries are not modified, while IIKW locations in non-pivot kernels are changed to zero, as shown in Figure 2(b). IS and VS arrays for non-pivot kernels k_0, k_2 are updated and IIKW stream is introduced to indicate locations of IIKW in the non-pivot kernels. Encoding for IIKW stream of non-pivot kernel weight is done with respect to pivot kernel weight, and is described in Table 3. The number of entries in IIKW stream is equal to number of non-zero kernel weights in the pivot kernel. Each entry in IIKW stream of a non-pivot kernel indicates how the pivot kernel weight is related to the non-pivot kernel weight. The encoding is done to obtain maximally compressed stream where single bit is sufficient for non-IIKW values, two bits are required for IIKW values and transition between them is demarcated implicitly using encoding of "01" as given in Table 3. Based on IIKW stream encoding for the non-pivot kernel, product generated using pivot kernel is broadcasted to adder associated with the non-pivot kernel with or without inversion. Thus multiplications for the IIKW in the non-pivot kernel are saved. We can observe that number of zeros in kernels k_0, k_2 have increased by an extra amount of 3,4 respectively due to IIKW search procedure as shown by grey cells in Figure 2(b). Generalizing the IIKW search procedure, ${}^{N}C_2$ comparisons are required for IIKW search in a group of N kernels to find one pivot kernel.

Given a CNN layer with M kernels and IIKW search procedure running on kernel group of N consecutive kernels, we explore the effect of size of kernel group on sparsity enhanced by IIKW. If N is equal to M, then IIKW search procedure produces a single pivot kernel for entire CNN layer. Sparsity enhancement due to single pivot kernel for an entire CNN layer is found to be very less. Further, we have experimented with values of $N = 4, 8, 16$ kernel groups per CNN layer (based on the fact that number of kernels in CNN

TABLE 4. CNN kernel sparsity enhanced due to IIKW on unpruned and pruned CNNs.

CNN	$N = 4$ Groups	$N = 8$ Groups	$N = 16$ Groups
Unpruned CNNs			
8-bit Precision			
VGG-16	4.69%	5.84%	6.87%
Inception-v3	4.13%	5.21%	6.14%
Inception-v4	5.60%	6.75%	7.86%
4-bit Precision			
VGG-16	9.31%	11.63%	13.83%
Inception-v3	11.54%	14.30%	16.73%
Inception-v4	11.15%	12.25%	13.32%
Pruned CNNs			
8-bit Precision			
VGG-16	3.68%	4.04%	5.24%
Inception-v3	1.2%	1.5%	3%
Inception-v4	1.17%	1.51%	1.8%
4-bit Precision			
VGG-16	4.48%	5.65%	6.80%
Inception-v3	9.18%	10.25%	11.33%
Inception-v4	7.65%	8.74%	9.84%

layers are generally a multiple of 16) and noted the sparsity enhancement. Each kernel group will have a pivot kernel and number of such kernel groups for a CNN layer is M/N, where M is the number of kernels in the layer. Table 4 shows the average sparsity enhancement on all layers due to IIKW for different sizes of kernel group on various unpruned and pruned state-of-the-art CNNs at 8-bit/4-bit precision. It can be observed that IIKW is able to improve CNN kernel sparsity across a wide range of state-of-the-art CNNs. Kernel group size of $N = 16$ is able to enhance kernel sparsity better than the other group sizes due to more kernels being available for comparison. We stopped at kernel group size of $N = 16$ since routing of the interconnect hardware for broadcasting to more than $N = 16$ non-pivot kernels becomes difficult to meet timing constraints. We can see that IIKW increases sparsity by at least 6% for 8-bit precision and at least 13% for 4-bit precision for all unpruned CNNs. For pruned CNNs, sparsity improvement is at least 1.8% for 8-bit precision and at least 6.80% for 4-bit precision. Since unpruned CNN models have more non-zero kernel weights for IKW comparison, we see that kernel sparsity due to IKW is more in unpruned CNN models compared to pruned CNN models. For 4-bit precision, sparsity enhancement is more than 8-bit because of just 16 unique values available for data representation. It can be concluded that IIKW can enhance sparsity across all combinations of quantized, unpruned or pruned CNN models, which makes it generic enough to be applied to various unpruned/pruned CNNs. We extend the concept of Identical IKW to Similar IKW which further enhances kernel sparsity, in Section II-E.

E. SIMILAR INTER-KERNEL WEIGHTS (SIKW)

We define SIKW as IKW which differ by a small magnitude with same or different signs. SIKW relaxes the constraint of exact match in magnitude and allows small differences between IKW for further enhancing the kernel sparsity. SIKW search procedure is same as IIKW search procedure of finding a *pivot kernel* and $(N-1)$ *non-pivot kernels* in a group of N kernels. The differences in magnitude considered for SIKW are:- 0,±1,±2,±4, since these differences translate to simple left shift operations during hardware realization. Figure 2(c) shows the pivot and non-pivot kernels after SIKW comparison. It can be observed that sparsity introduced due to SIKW is higher than IIKW, whose locations are indicated by SIKW stream. Encoding of the SIKW stream with respect to pivot kernel introduced in each non-pivot kernel is given in Table 5. In case of SIKW, product produced using pivot kernel is added/subtracted with IFM pixel used in pivot kernel to get the modified product, which is given to adders of non-pivot kernels. The generation of modified product to be broadcasted using IFM and product generated in pivot kernel is illustrated as follows: For example, consider co-ordinate location (0,2) of pivot kernel k_1 in Figure 2(c) having kernel weight of 11. When this kernel weight is multiplied with IFM, then the product generated corresponds to 11×IFM. But, the product required for non-pivot kernel k_0 is 9×IFM. Hence product generated using pivot kernel weight is subtracted by 2×IFM to obtain 9×IFM and broadcasted to adder of non-pivot kernel k_0. Thus, we can see that OFM produced using non-pivot kernel is exactly same as without using SIKW. Since differences considered are 0,±1,±2,±4, IFM pixel requires left shift and addition with product generated using pivot kernel to obtain the modified product.

TABLE 5. SIKW encoding for SIKW stream of Non-pivot kernel weight (y) with respect to Pivot kernel weight (x).

SIKW Encoding	Description	SIKW Encoding	Description
0	Not a SIKW	8	Not a SIKW
1	$y = x + 1$	9	$y = -(x + 1)$
2	$y = x + 2$	10	$y = -(x + 2)$
3	$y = x + 4$	11	$y = -(x + 4)$
4	$y = x$	12	$y = -x$
5	$y = x - 1$	13	$y = -(x - 1)$
6	$y = x - 2$	14	$y = -(x - 2)$
7	$y = x - 4$	15	$y = -(x - 4)$

Table 6 shows the sparsity enhancement due to SIKW for different sizes of kernel group on various unpruned and pruned state-of-the-art CNNs at 8-bit/4-bit precision. We can see that SIKW increases sparsity by at least 32% for 8-bit precision and 35% for 4-bit precision for unpruned CNNs. For pruned CNNs, SIKW increases sparsity by at least 9% for 8-bit precision and 18% for 4-bit precision. Similar to IIKW, SIKW also introduces higher sparsity in unpruned CNNs due to availability of more non-zeros in kernels. SIKW introduces significantly more sparsity than IIKW because SIKW is a super-set of IIKW. It can be concluded that SIKW can significantly improve sparsity in all scenarios and can even substitute pruning. Pruning is computationally intensive, since it involves re-training of the CNN. Hence pruning cannot be run on edge devices, due to their limited power and compute budget. IKW search operation is computationally less demanding

TABLE 6. CNN kernel sparsity enhanced due to SIKW on unpruned and pruned CNNs.

CNN	$N = 4$ Groups	$N = 8$ Groups	$N = 16$ Groups
Unpruned CNNs			
8-bit Precision			
VGG-16	27.6%	30.25%	34.97%
Inception-v3	26.87%	29.34%	32.58%
Inception-v4	31.65%	34.76%	37.54%
4-bit Precision			
VGG-16	32.67%	35.45%	38.77%
Inception-v3	37.60%	40.45%	43.22%
Inception-v4	28.9%	32.1%	35%
Pruned CNNs			
8-bit Precision			
VGG-16	25.6%	28.05%	31.13%
Inception-v3	10.3%	13.4%	16.24%
Inception-v4	5.62%	8.39%	9.56%
4-bit Precision			
VGG-16	12.75%	15.65%	18.87%
Inception-v3	26.89%	30.2%	33.57%
Inception-v4	24.25%	28.54%	32.77%

compared to pruning, since it does not involve training and is a single-pass operation. In Section III, we detail the proposed *IKW Architecture* which can utilize the enhanced sparsity due to IIKW or SIKW, and which can be introduced in well-known CNN accelerators.

III. IKW ARCHITECTURE

In the previous sections, we described how IIKW and SIKW increase sparsity in the kernels. The detected IIKW/SIKW will translate to improvement in performance and power when we have a supporting hardware architecture to utilize the IKW metadata. We target to come up with architectural elements that could be introduced in most of the state-of-the-art accelerators to support IKW computations. In this section, we propose a hardware architecture termed as *IKW Architecture*, which can work on IKW metadata, thus eliminating multiplication operation. Due to elimination of power hungry fixed-point multiplication, we expect reduction in power for MAC operation. This power saving is applicable to computation of every OFM pixel using non-pivot kernels. The proposed IKW Architecture can be introduced in any given CNN accelerator architecture, assuming that IFM pixel is convolved with multiple kernels in parallel, which is true for most CNN accelerator architectures.

Architecture supporting IKW needs to support transfer of product generated by the pivot kernel to adders corresponding to non-pivot kernels as specified by non-pivot kernel metadata. For realizing the IKW Architecture, the following two options are possible:

- to store the products generated by the pivot kernel in a local buffer, which is then read back later to be consumed by adders corresponding to non-pivot kernels.
- to broadcast the product generated by the pivot kernel as soon as it is produced and consumed by adders of non-pivot kernels.

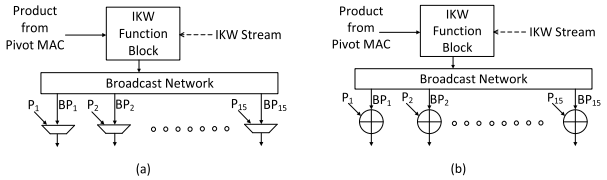


FIGURE 3. IKW Architecture template for CNN accelerator architecture (a) without kernel zero skipping feature consisting of 2:1 multiplexer to select between broadcasted product and product of non-pivot MAC (b) with kernel zero skipping feature containing adder to add broadcasted product and product of non-pivot MAC in the same cycle. Here $P_1, P_2, .., P_{15}$ corresponds to products generated by the non-pivot MAC, $BP_1, BP_2, .., BP_{15}$ corresponds to products broadcasted from the pivot MAC.

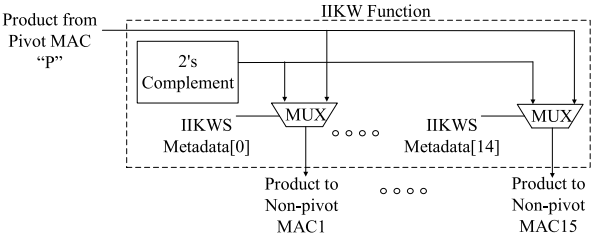


FIGURE 4. Function block for IIKW considering kernel group size of 16. It consists of 2's complement block and 2:1 multiplexer (MUX) block to select the product for each non-pivot MAC based on the IIKWS metadata.

In our approach, we chose the second option due to the following reasons:

- product generated by pivot kernel is needed only once by adders of non-pivot kernels per OFM pixel
- order of addition in non-pivot kernels can be out of order as long as the final OFM accumulated value is same as that without the IKW Architecture

Second option eliminates the need for local buffer and complex read back circuitry in each non-pivot kernel.

Figure 3(a),(b) show the proposed IKW Architecture templates for CNN accelerator architectures without and with kernel zero skipping feature respectively. Here, MAC working on pivot and non-pivot kernels are termed as *Pivot MAC* and *Non-pivot MAC* respectively. The three major components of the proposed IKW Architecture template are: IKW Function, broadcast network and adder/multiplexer before accumulator in Non-pivot MAC. Product generated by Pivot MAC is given to Function block. Function block transforms the product received from Pivot MAC as per IKW stream for each Non-pivot MAC as shown in Figure 4 for IIKW and Figure 5 for SIKW. IIKW Function block contains two's complement and 2:1 Multiplexer corresponding to every non-pivot MAC. SIKW Function block contains shifters, adders and 8:1 Multiplexer corresponding to every non-pivot MAC. Output of Function block is given to broadcast network supplying to each of the Non-pivot MACs. In each of the Non-pivot MACs, depending on whether the underlying CNN architecture uses kernel zero skipping or not, adder or multiplexer is inserted before accumulator.

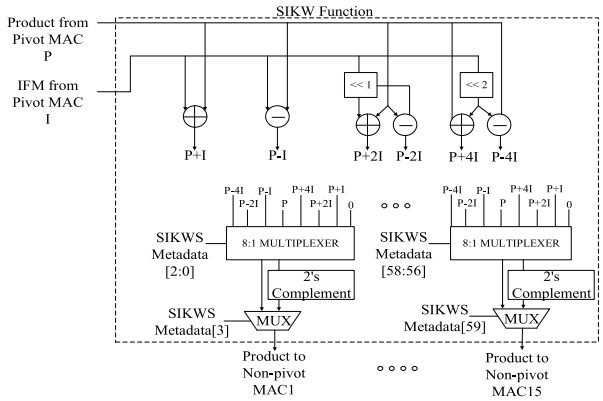
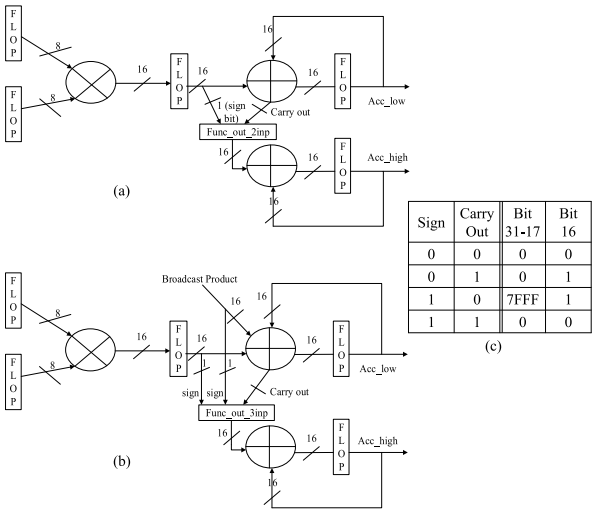


FIGURE 5. Function block for SIKW considering kernel group size of 16. It consists of add/subtract block, left shift block, 2's complement block and 8:1 multiplexer (MUX) block. Left shift block has to support shifts of 1 or 2. MUX block selects one of eight inputs for each non-pivot MAC based on SIKWS metadata.

The variation in IKW Architecture template depends on whether the underlying CNN accelerator architecture uses kernel zero skipping feature or not. In case where CNN accelerator architecture uses kernel zero skipping feature, then Pivot MAC and Non-pivot MACs are working at different rates due to kernels having different sparsity. In this case, the broadcasted product from Pivot MAC and product in Non-pivot MAC need to be added with the accumulated sum in the same cycle, which makes it necessary to have an additional adder in Non-pivot MAC. In case of kernels working in lock-step, product from Pivot MAC and product from Non-pivot MAC don't need to be added in the same cycle, hence multiplexer is sufficient in Non-pivot MAC. To reduce the switching activity in the accumulator registers, we discuss an optimization technique termed split-accumulator in III-A. This optimization is applied to accumulator in Pivot and Non-pivot MACs.

A. SPLIT-ACCUMULATOR OPTIMIZATION

Typical fixed-point MAC using 8 × 8-bit precision consists of multiplier producing 16-bit product which is then fed to a 32-bit accumulator. Before adding the 16-bit product to 32-bit accumulator, it is sign extended to 32 bits. Every addition involves updating all 32-bits of the accumulator register independent of magnitude of the product. To eliminate unnecessary updates to accumulator register bits, we have proposed an optimization called split-accumulator optimization. Figure 6(a) shows the split-accumulator optimization, where accumulator register is divided into two equal parts of 16-bit each (Acc_high,Acc_low). The lower 16-bits of accumulator are updated every cycle since it is directly affected by the incoming product bits. The upper 16-bit update is a function of carry out bit of lower 16-bit addition and sign of the product as obtained using logic table in Figure 6(c). It can be observed that upper 16-bits of accumulator register needs to be updated in two out of four cases, as shown in Figure 6(c), which leads to lesser updates in upper 16-bit accumulator register.



Sign	Carry Out	Bit 31-17	Bit 16
0	0	0	0
0	1	0	1
1	0	7FFF	1
1	1	0	0

(c)

FIGURE 6. (a) Two input split-accumulator optimization. "Acc_high" accumulator flop is updated if output of "Func_out_2inp" is non-zero. (b) Three input split-accumulator optimization. "Acc_high" accumulator flop is updated if output of "Func_out_3inp" is non-zero. (c) Func_out_2inp truth table for updating upper 16-bits of two input accumulator. Bits [31-17,16] together form the 16-bit output from Func_out_2inp block. Func_out_2inp gives non-zero output in two out of four cases, thus reducing update frequency of Acc_high.

TABLE 7. Func_out_3inp of three input split-accumulator. *X* indicates don't care, Bits [31-18,17,16] together form the 16-bit output of Func_out_3inp block of Figure 6(b).

Carry out	Sign of Broadcast Product	Sign of Product	Bit 31-18	Bit 17	Bit 16
2'h0	0	0	14'h0	0	0
2'h0	0	1	14'h0	0	1
2'h0	1	0	14'h0	1	0
2'h0	1	1	14'h3FFF	1	1
2'h1	0	0	14'h0	0	0
2'h1	0	1	14'h0	0	1
2'h1	1	0	14'h3FFF	1	1
2'h1	1	1	14'h0	0	0
2'h2	0	0	14'h0	0	0
2'h2	0	1	14'h3FFF	1	1
2'h2	1	0	14'h3FFF	1	0
2'h2	1	1	14'h0	0	1
2'h3	*X*	*X*	14'h0	0	0

In case of three input accumulator, the addition of three 16-bit inputs can produce 2-bit carry out. Table 7 details the output which needs to be added to upper 16-bit accumulator as function of 2-bit carry out from lower 16-bit 3-input addition and sign extension of non-pivot MAC product and broadcasted product from pivot MAC. Figure 6(b) shows the split-accumulator optimization for three input adder where upper 16-bits of accumulator is updated based on the value of ''Func_out_3inp''. Section III-B details the RTL implementation and evaluation methodology used for obtaining the execution performance and power numbers due to IKW Architecture on state-of-the-art CNN accelerator architectures.

B. RTL IMPLEMENTATION AND EVALUATION

For evaluating the improvement in power and execution performance due to IKW Architecture on three

state-of-the-art CNN accelerator architectures (NVDLA, Cnvlutin and ZeNA), we describe the evaluation methodology used in detail. The choice of CNN accelerator architectures covers three variations with respect to usage of IFM/kernel sparsity in the architecture. NVDLA uses zero valued IFM/kernel to switch off the multiplication operation in MAC, but does not use it for improving execution performance by zero skipping. Cnvlutin uses zero values of IFM to improve execution performance by zero skipping the computation and performing only computation corresponding to non-zero values of IFM. ZeNA skips computation if either of IFM or kernel values are zero. Thus, the three architectures have three different variations with respect to usage of IFM/kernel values for accelerating convolution operation. We could not find any well-known state-of-the-art architecture using only kernel zero skipping to improve performance. Hence, impact of IKW Architecture introduction is shown for three scenarios only. Integrating IKW Architecture to accelerator architecture using only kernel zero skipping to improve performance is similar to ZeNA, since IKW is only dependent on kernels.

We implemented the RTL designs of the three CNN accelerator architectures at 8-bit and 4-bit precision. These RTL implementations are referred to as *Baseline Architecture*. The three baseline designs were augmented with IKW Architecture templates and coded in RTL at 8-bit and 4-bit precision. Along with IKW Architecture template, split-accumulator optimization was applied to Pivot MAC and Non-pivot MAC. Baseline architecture augmented with IKW Architecture and split-accumulator optimization is referred to as *Enhanced Architecture*. Enhanced Architectures were created for IIKW and SIKW separately to study their effect on area, power and performance. A bit-accurate Python model was built for Baseline and Enhanced Architectures for generating the reference OFM pixels. The IFM and kernel values from pruned VGG-16, Inception-v3/v4 were extracted using Caffe tool at 8-bit/4-bit precision and given to Python script and RTL designs. We have used pruned CNNs for power estimation because sparsity enhanced due to IKW is lesser in them. We wanted to check the minimum possible power savings due to IKW. Baseline and Enhanced Architectures using IFM and kernel values from Caffe were simulated using Synopsys VCS tool [43] and OFM pixels were compared with Python reference model to check correctness of RTL implementation. During VCS simulation, cycles taken by Baseline and Enhanced Architectures to complete execution of CNN layers were noted for measuring the performance improvement due to IKW Architecture.

Functionally correct RTL implementations of Baseline and Enhanced Architectures were synthesized using Synopsys Design Compiler tool [44] with Samsung 10 nm technology node and 800 MHz target frequency to generate the netlist, and area numbers were noted. SRAM database files at Samsung 10 nm technology node were generated using Synopsys Library Compiler [44] and used in synthesis. The netlist generated was used for power estimation using Spyglass Power Estimation tool [45] to get module level power consumption. The input to Spyglass Power Estimation tool consisted of netlist generated after synthesis, activity factor file generated using VCS simulation, database of memory files and technology libraries. Power estimation tool gives average power consumed for the design within a specified time period. We report average power consumed per different layers of CNN, simulated for the three state-of-the-art CNN accelerator architectures. The above procedure is used for measuring the impact of IKW Architecture in three state-of-the-art CNN accelerator architectures. In the following sections, we augment state-of-the art CNN accelerator architectures like NVDLA [18], Cnvlutin [17] and ZeNA [19] with IKW Architecture and demonstrate power and performance improvement. In section III-F, we discuss about the results obtained and contrast it with UCNN approach.

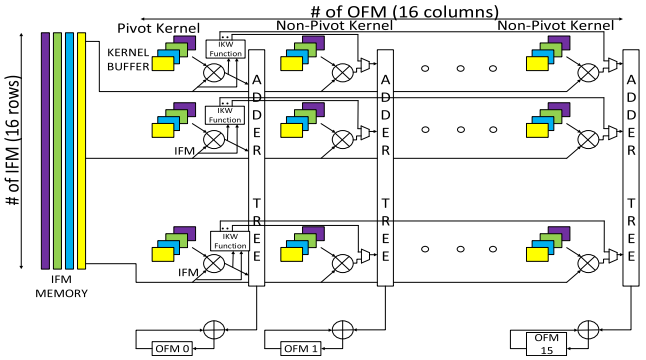


FIGURE 7. NVDLA like architecture augmented with IKW Architecture.

C. IKW ON NVDLA

NVDLA architecture [18] proposed by Nvidia, is an example of dense CNN accelerator architecture, which does not use IFM/kernel zero skipping for enhancing execution performance. It stores all data (IFM, OFM and kernels) in channel first storage format i.e. pixels/kernel weights belonging to same planar co-ordinates but different channels are packed together in a single location. Compute unit consists of IFM shared with multiple kernels executing in parallel as shown in Figure 7, where each column consists of multipliers, adder tree and accumulator working on a single kernel at 8-bit precision. We have implemented baseline NVDLA like architecture with 16 kernels working in parallel sharing single IFM. Each memory location stores 16 elements in channel direction at 8-bit precision and 32 elements at 4-bit precision. 16 kernels are stored in 16 kernel buffers and IFM is supplied from IFM memory. Each column consists of 16 multipliers, whose outputs are given to an adder tree, which feeds an OFM pixel accumulator.

Figure 7 shows Enhanced NVDLA like architecture augmented with IKW Architecture to support IKW detected in kernels. Along with addition of IKW Architecture, OFM accumulators are optimized using 2-input split accumulator technique for IIKW and 3-input split accumulator technique

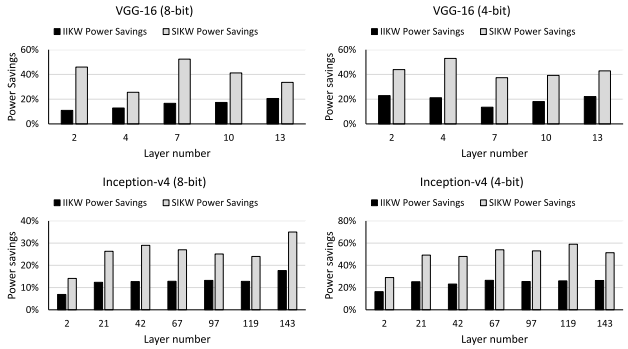


FIGURE 8. Power savings for NVDLA like architecture augmented with IKW Architecture on pruned CNNs.

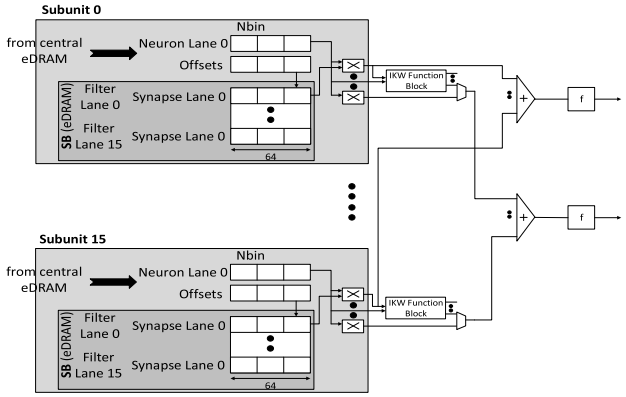


FIGURE 9. Cnvlutin architecture augmented with IKW Architecture.

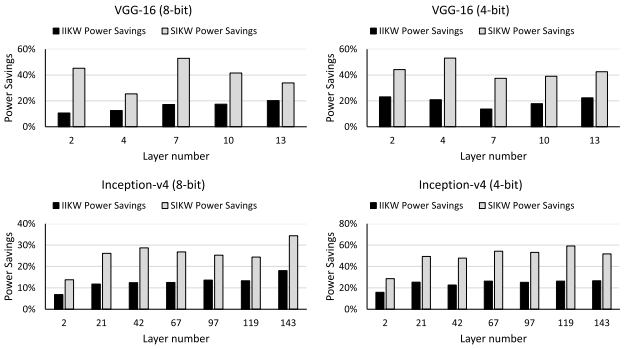


FIGURE 10. Power savings for Cnvlutin architecture augmented with IKW Architecture on pruned CNNs.

for SIKW. An extra kernel memory is introduced for storing the IKW metadata. IKW Architecture template used is from Figure 3(a) since NVDLA does not use kernel zero skipping feature. The first column consists of pivot kernel, while remaining columns contain non-pivot kernels. The product from each multiplier of first column is given to IKW Function block, which transforms the product as per IKW stream of non-pivot kernels and broadcasts it to adders in the same row. Note that each row of multipliers work on the same IFM/kernel channel and hence broadcast of products is along the row. Multiplexer present before adder in the non-pivot kernel column selects between broadcasted product from pivot kernel or product from it's own kernel. Since all kernels move in lock-step, there is never a possibility of both the inputs of multiplexer (MUX) being valid and non-zero at the same cycle. IKW Architecture reduces power consumption in NVDLA like architecture, but does not change execution performance.

Area, Power estimation of RTL implementation of Baseline NVDLA like architecture and Enhanced NVDLA like architecture was done as explained in methodology Section III-B. Area increase for Enhanced Architecture with IIKW support is around 6.9% for both 4-bit and 8-bit precision. Area increase for Enhanced Architecture with SIKW support is around 9.2% for both 4-bit and 8-bit precision. We see that area increase is not significant compared to Baseline NVDLA like architecture. Figure 8 shows the power savings due to IKW Architecture over the baseline NVDLA architecture. We observe power savings of at least 14% for 8-bit and at least 29% for 4-bit due to SIKW for VGG-16 and Inception-v4 CNNs. In the next section, we map IKW Architecture to Cnvlutin CNN accelerator which uses IFM zero skipping unlike NVDLA.

D. CNVLUTIN

Cnvlutin [17] architecture, proposed by Albericio et. al., is an example of CNN accelerator architecture, which uses IFM zero skipping for enhancing execution performance. Cnvlutin is similar to NVDLA, except IFM is decoupled into 16 different streams so that IFM zero skipping can proceed independently in each of the streams. Each stream corresponds to separate channel and hence the product after multiplication can be added using adder tree. The IFM zero skipping is handled by using encoding IFM in Zero-Free Neuron Array Format that enables it to skip zero valued IFM computations and supply only non-zero IFM values to multipliers. The kernel weights are chosen based on the offset for each non-zero IFM pixel. The back-end part of 16 multipliers, adder tree and OFM accumulator per kernel is same as NVDLA.

Figure 9 shows the Cnvlutin architecture augmented with IKW Architecture template (Figure 3(a)). Integration of IKW Architecture to Cnvlutin is very similar to NVDLA, except that IKW stream is addressed by the offset of IFM. Offset is used to index the kernel weight as well as IKW stream to decide to which non-pivot kernel, the product generated by pivot kernel needs to be broadcasted to. IKW Architecture is introduced in each Subunit, where Filter Lane 0 is the pivot kernel in all of the subunits. Filter Lanes 1 - 15 correspond to non-pivot kernels.

Area, Power estimation of RTL implementation of Baseline Cnvlutin architecture and Enhanced Cnvlutin architecture were done as explained in methodology Section III-B. Area increase for Enhanced Architecture with IIKW support is around 6.9% for both 4-bit and 8-bit precision. Area increase for Enhanced Architecture with SIKW support is around 9.2% for both 4-bit and 8-bit precision. We see

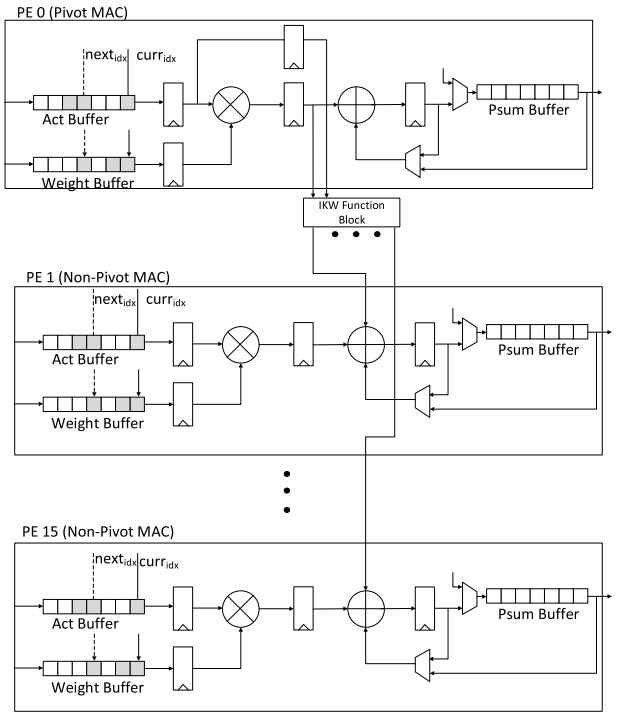


FIGURE 11. ZeNA architecture consisting of 16 parallel processing elements (PE) consisting of MAC units working on sixteen kernels in parallel producing 16 OFMs. Kernel running in PE0 is Pivot kernel which broadcasts the product to remaining 15 PEs. Proposed IKW Architecture consists of IKW Function block and broadcast network. IKW Function block can be IIKW or SIKW. IIKW function block uses only product from Pivot MAC to generate products to be broadcasted to non-pivot MACs. SIKW function block uses product and IFM from pivot MAC to generate products to be broadcasted to non-pivot MACs.

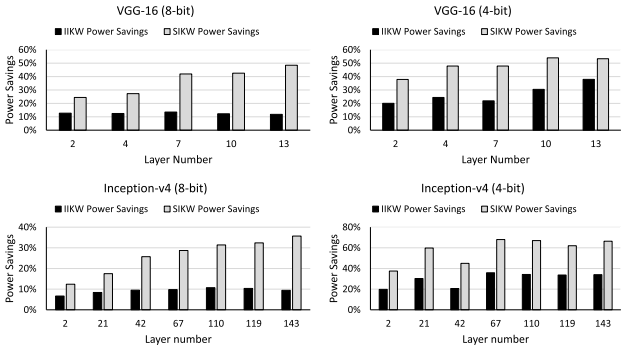


FIGURE 12. Power savings for ZeNA architecture augmented with IKW Architecture on pruned CNNs.

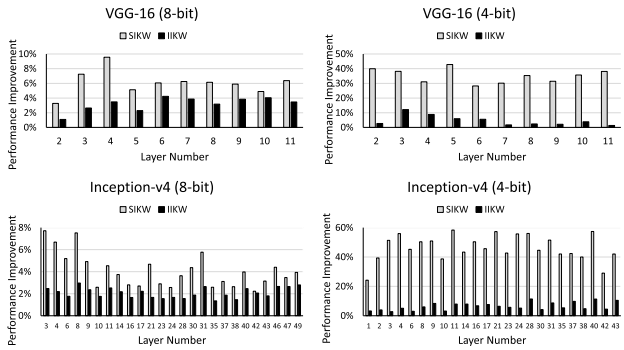


FIGURE 13. Performance improvement in percentage of cycles in ZeNA architecture due to IIKW and SIKW on unpruned CNNs.

that area increase is not significant compared to Baseline Cnvlutin architecture. Figure 10 shows power savings due to IKW Architecture for VGG-16 and Inception-v4 CNNs. We observe power savings of at least 13% at 8-bit and at least 28% at 4-bit due to SIKW.

E. ZeNA

ZeNA [19] architecture, proposed by Kim et. al., is an example of CNN accelerator architecture, which uses both IFM and kernel zero skipping for enhancing execution performance. In ZeNA CNN architecture, each processing element (PE) unit has its own IFM buffer and kernel buffer so that MAC unit execution time is independent of other MAC units. Since each MAC unit has its own IFM and kernel zero skipping logic, each PE completes OFM execution at a different rate compared to other PEs. To counter this rate imbalance within PEs and across PEs, authors have proposed zero-aware kernel allocation and work stealing approaches. Work stealing approaches deal with PEs which have completed their work and are borrowing work from PEs which still have work remaining. Zero-aware kernel allocation rearranges kernel channels according to increasing density and distributes them inside PE clusters working on the same kernel.

To map IKW on the ZeNA architecture, considering zero-aware kernel allocation and work stealing approaches, IKW search process is applied to find pivot and non-pivot kernels before organizing kernel channels as per density. Since IKW search process affects kernel density by introducing zeros in non-pivot kernel, IKW search process has to be run before kernel allocation. During zero-aware kernel allocation, along with kernel channel, IKW stream for the corresponding channel are also rearranged. Since order of accumulation of OFM is not important as long as final OFM value remains unchanged, IKW can be applied to zero-aware kernel allocation process. In case of work stealing approach, the restriction due to IKW is that PE working on non-pivot kernel cannot steal work from PE working on pivot kernel. This is because broadcast network can only send data from PE working on pivot kernel to other PEs. If PEs working on non-pivot kernel steal work from pivot kernel, they cannot broadcast products to other PEs.

A single cluster of ZeNA architecture without the work-stealing logic, considered as Baseline for evaluation, was implemented in RTL. The Baseline Architecture was then augmented with IKW Architecture at the level of PE as shown in Figure 11. IKW Architecture template shown in Figure 3(b) is used, which contains an adder before the accumulator, since the underlying CNN architecture uses kernel zero skipping feature. Area, Power estimation of RTL implementation of Baseline ZeNA architecture and Enhanced

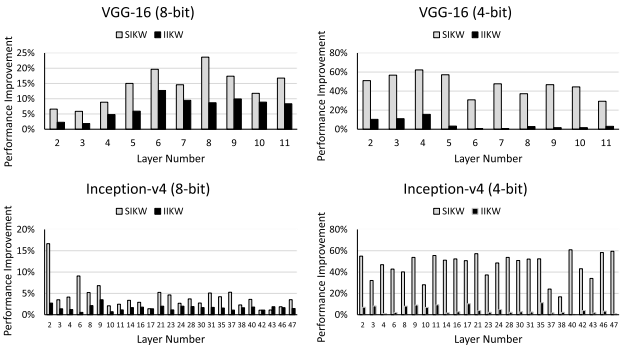


FIGURE 14. Performance improvement in percentage of cycles in ZeNA architecture due to IIKW and SIKW on pruned CNNs.

ZeNA architecture were done as explained in methodology Section III-B.

Area increase for Enhanced Architecture with IIKW support is around 9.5% for both 4-bit and 8-bit precision. Area increase for Enhanced Architecture with SIKW support is around 14.7% for both 4-bit and 8-bit precision. We see that area increase is not significant compared to Baseline ZeNA architecture. Figure 12 shows the power savings due to IKW Architecture over the Baseline ZeNA architecture for VGG-16 and Inception-v4 CNNs. We observe power savings of at least 12% for 8-bit and at least 19% for 4-bit due to SIKW. Further, IKW Architecture enhances execution performance due to more kernel skipping opportunities in non-pivot kernel compared to original kernel. Performance improvement is limited by the slowest kernel among a set of pivot and non-pivot kernels. Figure 13 shows the performance improvement in percentage of cycles in ZeNA for unpruned VGG-16 and Inception-v4 CNNs. Figure 14 shows the performance improvement in percentage of cycles in ZeNA for pruned VGG-16 and Inception-v4 CNNs. We observe performance improvement of at least 2% for 8-bit and 13% for 4-bit across VGG-16 and Inception-v4 CNNs. Since IKW improves execution performance and reduces power consumption, it gives double benefits for architectures using kernel zero skipping feature.

F. DISCUSSION

In the previous sections, we have seen IKW Architecture templates introduced in well-known CNN accelerator architectures for improving power efficiency and execution performance of CNNs. IKW Architecture was able to deliver power savings across multiple fixed-point bit precision (8-bit, 4-bit) formats, and execution performance improvement in case of ZeNA architecture. We can conclude that power savings validate the idea that power saved due to fixed-point multiplication is more than power spent in IKW Function block, broadcast network and adder/multiplexer in non-pivot MAC. The amount of power savings achieved per layer is a function of sparsity enhanced due to IKW. We have observed that kernels with higher enhanced sparsity due to IKW deliver more power savings. Next, we comment on extending SIKW concept using different relation between kernel weights, introduction of IKW Architecture on systolic array architecture and impact of IKW Architecture on power efficiency of the CNN accelerator architecture.

The concept of SIKW can be extended to other possible relations between the kernel weights. For example, if kernel weight is half or $2\times$ or $4\times$ the other kernel weight, product generated using pivot kernel needs to be right-shifted by 1 or left-shifted by 1 or 2 respectively, to get the value which can be broadcasted for accumulator working on non-pivot kernel. If the relation between kernel weights allows a cost-effective way to compute the product which needs to be broadcasted, then such a relation will result in power savings using IKW.

We address introduction of IKW Architecture in systolic array architecture [46], where there is a tight temporal alignment between arrival of IFM and kernel at each Processing Element (PE). As described by Kwon *et al.* [46], a typical systolic array architecture consists of 2-D grid of PEs, where each PE performs MAC operation for computation of OFM pixels. In the 2-D grid of PEs, each PE is connected to four neighbouring PEs, except at the grid boundaries. IFM is streamed horizontally to PEs starting from left to right, kernels are streamed to PEs from top to bottom. OFMs are collected from the bottom row of the grid. Kernels are shared column wise, while IFMs are shared row wise. A single PE row works on the same IFM window, with columns computing different OFM channels. Different IFM window is given to each PE row. IKW Architecture is applied to every row of PEs, where pivot kernel can be mapped to leftmost column of 16 PEs, with remaining 15 columns working on non-pivot kernels. If the number of columns exceed 16, then total columns are divided into groups of 16 and IKW Architecture is applied to every set of 16 columns. IKW Architecture shown in Figure 3(b) is used here, although systolic array architecture does not use kernel zero skipping feature. It is because PEs in the same row receive IFM pixels in a staggered manner and PEs working with non-pivot kernel need to add broadcasted product received from PE working on pivot kernel with their own products in the same cycle. This necessitates the usage of IKW Architecture of Figure 3(b). It can be observed that completion of OFM computation happens in a staggered manner, with PEs closer to IFM source finishing earlier than PEs away from it. This leads to scenario where PE working on pivot kernel finishes OFM pixel computation earlier compared to remaining 15 PEs working on non-pivot kernels in a group of 16 columns. Now, PE working on pivot kernel starts broadcasting product corresponding to next OFM pixel while remaining PEs are working on previous OFM pixel. To support this scenario, an extra accumulator is required in each of the PEs working on non-pivot kernels. Summarizing, the IKW Architecture required for supporting IKW in systolic array architecture consists of architecture template of Figure 3(b) with an extra accumulator in each of the PEs working on non-pivot kernels.

We comment about the impact on power and energy efficiencies due to introduction of IKW Architecture in three state-of-the-art accelerator architectures, namely NVDLA, Cnvlutin and ZeNA. In case of architectures not using kernel zero skipping like NVDLA/Cnvlutin, proposed IKW Architecture does not modify cycles required to complete the CNN layer. However, it gives power savings of at least 13% for 8-bit case. We noticed improvement in power efficiency which is mainly due to the power savings of at least 13%, since the total average operations per second executed is similar in baseline and baseline with IKW Architecture. In case of architectures using kernel zero skipping like ZeNA, proposed IKW Architecture improves the execution performance by at least 2% and reduces power consumption by at least 12% for 8-bit case. We observe improved power efficiency due to power savings similar to architectures not using kernel zero skipping. Since, there is at least 2% improvement in execution performance, we get better energy efficiency compared to baseline architecture. Energy efficiency helps in improving battery life, which is especially useful for edge devices, which needs to complete tasks faster and consume less active power. Next, we contrast our proposal to UCNN [36], which explored identical kernel weights within a kernel i.e. intra-kernel weights.

In UCNN approach, kernel weights within a kernel are compared to generate metadata containing information about repeated kernels. During dot product operation, products corresponding to repeated kernels are stored in memory. The stored products are retrieved from the memory if the product of IFM pixel to be multiplied with repeated kernel is already present. The product is retrieved using indirection pointers using the IFM pixel value. Compared to IKW approach, in UCNN, the IFM pixel has to be compared during runtime to access products stored in memory. Also, it requires memory and indirection pointer to store the repeated products. Due to presence of memory, the hardware architecture is expensive in terms of area and power. If UCNN is applied to 4-bit precision, it will result in significant memory accesses to fetch products because of higher number of repeated IFM pixel values. The approach followed in IKW, where the products are used as soon as possible, is less area intensive, without need for IFM pixel value comparison. In addition, as the number of kernel channels increase, opportunities to introduce sparsity in IKW is more compared to UCNN.

IV. CONCLUSION

Given a trained, pruned or unpruned, quantized Convolutional Neural Network (CNN) model, we have shown that the proposed Inter-Kernel Weight (IKW) technique can be applied to 3-D Convolution layers or Fully Connected layers to extract identical and/or similar inter-kernel weights to eliminate redundant multiplication operations. Proposed IKW technique can be applied to other types of 3-D Convolution operators like Dilated Convolution [47], Transposed Convolution [48] as well, which increases its applicability to a wide variety of CNNs used in other application domains such as monocular depth estimation, scene segmentation, etc. From the results, we see that the proposed IKW technique using Similar Inter-Kernel Weights (SIKW) introduces at least 30% sparsity enhancement in case of unpruned state-of-the-art CNNs and at least 10% sparsity enhancement in pruned CNNs. Since IKW can give approximately 30% sparsity enhancement on unpruned CNNs, it can be considered as a candidate to replace pruning. Pruning is required for reducing the size of given unpruned model to improve power and execution performance. Pruning is not possible on edge devices, because it is a resource intensive step, requiring re-training. Instead, IKW search procedure is less power hungry and can be done on edge devices to generate a sparse CNN model.

Further, to efficiently use the IKW detected in CNNs, we have proposed a hardware architecture template called *IKW Architecture* which can be introduced in any state-of-the-art CNN accelerator. IKW Architecture eliminates redundant fixed-point multiplications, thus saving power. It can also give improvement in performance if CNN accelerator uses kernel zero skipping feature, similar to ZeNA. Proposed IKW Architecture using SIKW gives power savings of at least 12% for 8-bit precision and 19% for 4-bit precision on well known CNN accelerator architectures, for state-of-the-art CNNs, with a maximum area overhead of 14.7%. It also improves performance by at least 2% for 8-bit precision and 13% for 4-bit precision for ZeNA. Thus, the proposed IKW Architecture enables power efficient CNN inference on edge devices by reducing power consumption of 3-D Convolution operations. We consider IKW technique as an alternative approach for enhancing CNN kernel sparsity, which can be run on edge devices. It can be concluded that the proposed IKW technique and architecture is a valuable power efficient solution for enhancing edge computing.

REFERENCES

[1] 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. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 2818–2826.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, p. 248.

[3] 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.*, Jun. 2014, pp. 580–587.

[4] 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*. [Online]. Available: https://arxiv.org/abs/1312.6229

[5] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places Database," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, Cambridge, MA, USA, 2014, pp. 487–495.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1725–1732.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

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

[11] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," 2014, *arXiv:1412.5567*. [Online]. Available: https://arxiv.org/abs/1412.5567

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[13] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, New York, NY, USA, 2014, pp. 269–284.

[14] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[15] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2015, pp. 92–104.

[16] 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.

[17] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[18] NVDLA. (2018). *The NVDLA Deep Learning Accelerator*. [Online]. Available: http://nvdla.org/hw/v1/ias/unit_description.html

[19] D. Kim, J. Ahn, and S. Yoo, "ZeNA: Zero-aware neural network accelerator," *IEEE Des. Test. Comput.*, vol. 35, no. 1, pp. 39–46, Feb. 2018.

[20] Y. L. Cun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage*. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[21] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe, "Optimal brain surgeon: Extensions and performance comparisons," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst.*, San Francisco, CA, USA, 1993, pp. 263–270.

[22] R. Reed, "Pruning algorithms–A survey," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.

[23] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–18, May 2017.

[24] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, Cambridge, MA, USA, 2015, pp. 1135–1143.

[25] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CORR*, vol. abs/1611.06440, pp. 1–17, Oct. 2016.

[26] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient Integer-Arithmetic-Only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[27] Y. Wu and C. T. Huang, "Efficient dynamic fixed-point quantization of CNN inference accelerators for edge devices," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2019, pp. 1–4.

[28] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 2849–2858.

[29] P. Judd, J. Albericio, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," *CoRR*, vol. abs/1511.05236, pp. 1–12, 2015.

[30] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 3009–3018.

[31] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*. [Online]. Available: https://arxiv.org/abs/1806.08342

[32] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–8.

[33] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and Internet-of-Things devices," in *Proc. Int. Workshop Internet Things Towards Appl.*, 2015, pp. 7–12.

[34] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. ICANN*, 2014, pp. 281–290.

[35] D. Kim, J. Ahn, and S. Yoo, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1462–1467.

[36] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "UCNN: Exploiting computational reuse in deep neural networks via weight repetition," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 674–687.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[38] 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.

[39] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010, pp. 253–256.

[40] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CORR*, vol. abs/1311.2901, pp. 1–11, Jul. 2013.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*. [Online]. Available: http://arxiv.org/abs/1408.5093

[43] (2019). *VCS Compiler and Simulator*. [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html

[44] (2019). *Design Compiler*. [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx

[45] Synopsys. (2019). *Spyglass Power*. [Online]. Available: https://www.synopsys.com/verification/static-and-formal-verification/spyglass/spyglass-power.html

[46] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2018, pp. 461–475.

[47] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, pp. 1–13, Feb. 2015.

[48] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, *arXiv:1603.07285*. [Online]. Available: http://arxiv.org/abs/1603.07285

PRAMOD UDUPA (Senior Member, IEEE) received the B.E. degree in electronics and communication engineering from Visvesvaraya Technological University, India, in 2006, the M.E. degree in microelectronics from BITS Pilani, Pilani, India, in 2008, and the Ph.D. degree in VLSI for digital signal processing from INRIA, France, in 2014.

From 2014 to 2016, he has worked on 4G systems as a Design Engineer in Bengaluru, India. From 2016 to 2018, he also worked as an IP Logic Design Engineer at the Intel India Development Center, Bengaluru. Since 2018, he has been working as a Staff Engineer with the Samsung Advanced Institute of Technology, SRI-B, Bengaluru. His research interests include low power algorithms and architectures for machine learning applications, communication systems involving OFDM systems. He has published articles in conferences, such as ICC, VTC, and ISCAS.

GOPINATH MAHALE received the B.E. degree in electronics and communication engineering from Visvesvaraya Technological University, India, in 2007, the M.Tech. degree in electronics from the University of Pune, India, in 2011, and the Ph.D. degree in electronic systems engineering from the Indian Institute of Science, in 2017.

He has worked as a Project Engineer at Wipro Technologies, from 2007 to 2009, a Research Associate at the Indian Institute of Science, from 2016 to 2017, and a Postdoctoral Research Associate at the University of Paderborn, Germany, from 2017 to 2018. Since April 2018, he has been working as a Staff Engineer at the Samsung Advanced Institute of Technology, SRIB, Bengaluru, India. His research interests include domain specific hardware accelerators, deep learning, low power computations for deep learning, and image processing.

KIRAN KOLAR CHANDRASEKHARAN received the B.E. degree in electronics and communication engineering from the Peoples Education Institute of Technology, Bangalore University, Bengaluru, India, in 2000.

From 2004 to 2012, he was the Project Lead with Sanyo Semiconductor Company, Ltd., Japan. Since 2013, he has been a Staff Engineer with the Samsung Advanced Institute of Technology, SRIB, Bengaluru. His research interests include convolutional neural network hardware accelerator architecture and GPU architectures.

SEHWAN LEE received the B.S. degree and the M.S. degree in electrical Engineering from Yonsei University, South Korea.

He joined Samsung Electronics Company, Ltd., in 2010, where he has delivered successfully a lot of video and image IP and SoC products in important projects. Since 2015, he has been working as a Principal Researcher on the neural network accelerator and neuromorphic system with the Samsung Advanced Institute of Technology, South Korea. His research covers a broad range of topics related to deep learning algorithms, video processing algorithms, neural network acceleration, and general processor architectures.