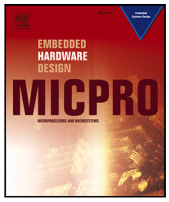
Microprocessors and Microsystems 98 (2023) 104776

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



Adaptive pooling-based convolution factorization for deploying CNNs on energy-constrained IoT edge devices

Amin Yoosefi [a,*], Mehdi Kargahi [a,b]

[a] School of ECE, College of Eng., University of Tehran, Iran
[b] School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

ARTICLE INFO

Keywords:
Convolutional neural network (CNN)
Resource-constrained edge devices
Edge computing
Convolution factorization
IoT

ABSTRACT

Convolutional neural networks (CNNs) involve a tremendous amount of multiply-and-accumulate (MAC) computations, leading to energy issues when deployed on energy-constrained IoT edge devices for computer vision purposes. However, a great number of these computations are ineffectual since they lead to meaningless zero activations, which could be avoided if predicted appropriately. This has been the focus of many studies, whereas it has only been done through zero-activation simplifications. This paper goes beyond those simplifications through a convolution factorization approach to convert the MAC-intensive convolution operations into less complex pooling operations. The main characteristic of the proposed idea is that, given a target accuracy, the applicability and type of factorization are adaptively decided for each individual activation. This is achieved by providing a lightweight multi-class CNN-based predictor, namely RedZAP. The selected types of pooling operations for our convolution factorizations are supported by existing CNN architectures. The experimental results show that, given a top-1 accuracy loss constraint of 1%, the proposed approach saves another 8% MAC operations in comparison to the existing zero-prediction approaches in the literature.

1. Introduction

Machine learning (ML) algorithms are widely used in various pattern recognition and data mining applications to enable data analysis on IoT edge devices, such as IoT vision sensor nodes, for massive volumes of data generated at the network edge [1]. Convolutional neural networks (CNNs), known as an instance of ML algorithms and, more precisely, deep neural networks(DNNs), have proven a significant order of superiority over other ML instances by providing high levels of accuracy in such applications [2]. As an example, CNNs have demonstrated beyond human-level accuracy in computer vision tasks, such as image classification and object detection [3]. This superior performance of CNNs comes at the expense of high computational demands, thus leading to energy inefficiencies and hindering their practical applications on edge devices with strict energy constraints. These edge constraints range from energy limitations of battery-operated smartphones (with mobile processors) to even more extreme cases such as small IoT sensor nodes (with low-power processors and energy harvesters).

Cloud computing can effectively satisfy the high computational demands of CNNs by having the CNN computations offloaded to more powerful, resource-rich servers, called *the cloud*. However, on its own, Cloud computing cannot be a viable solution, due to several issues mostly coming from the *long-distance communication network* between the cloud and edge devices. First, the delay associated with the long-distance network can potentially threaten the timeliness requirements of time-sensitive applications with real-time nature, such as robotics, self-driving car, and augmented reality [4]. Second, privacy and security concerns may arise when sensitive data, such as patient-generated health data in healthcare applications, is exposed to potential unauthorized access as a result of the data offloading. Finally, network connectivity can be of major concern in circumstances where access to the central node is limited or transient, e.g., in oil fields. All these issues with the long-distance network imply the idea of performing the data processing in close proximity to data sources (e.g., sensors) on energy-constrained edge devices, introducing a new paradigm called *edge computing*.

With edge computing, CNN computations are migrated from the cloud to the edge of the network. The applicability of this migration is restricted by energy limitations in deploying computationally complex CNNs with high energy demands on energy-constrained edge devices. This high computational complexity is defined in terms of a tremendously significant amount of multiply-and-accumulate (MAC) computations involved in the so-called *convolutional (CONV) layers*, which enable high levels of accuracy in CNNs by providing high-level *feature-extraction* capabilities through learning arrays of weights, called

* Corresponding author.
E-mail addresses: a.yoosefi@ut.ac.ir (A. Yoosefi), kargahi@ut.ac.ir (M. Kargahi).

https://doi.org/10.1016/j.micpro.2023.104776
Received 1 September 2020; Received in revised form 3 March 2021; Accepted 22 January 2023
Available online 3 February 2023

kernels. As demonstrated by [5], AlexNet [6] and VGG-16 [7], as two variants of a CNN, require approximately 666M and 15.4B MAC operations, respectively, just for a single image classification task. All these considerations clarify the importance of energy-efficient processing of CNNs on devices with resource limitations and define the huge amount of involved MAC operations as the potential challenge ahead to be addressed.

The inherent data redundancy existing in CNNs is usually exploited to mitigate the huge MAC operations overhead. In CNNs, an *output activation* is a weighted sum of a collection of spatially adjacent *input activations*, sharing very close values. In other words, the value of an output activation depends on the aggregation of a wide set of input activations with intrinsic redundancy, rather than on individual activations. Due to this redundancy, errors on some input activations are mitigated by the correct redundant activations in the set, applying a self-healing property to the application.

An extremely wide literature pertains to MAC reductions in CNNs, exploiting the inherent data redundancy characteristic. The studies commonly follow five lines of research: (i) proposing new DNN model designs with an extremely reduced amount of parameters and MAC operations [3,4,8–18]; (ii) automatically searching for a neural architecture with the best possible performance, referring to a research field known as neural architecture search (NAS) [19–25]; (iii) compressing existing DNN model designs [26–50]; (iv) accelerating DNN model designs in hardware [51–57]; and (v) predicting zero-valued activations and skipping them [58–68]. This paper is very closely related to the studies taking a prediction approach.

The studies employing zero-prediction techniques exploit the empirical observation that a considerable amount of output activations in a CONV layer yield meaningless zero values. According to [67], on average, 65.67% and 47.10% of total pixels result in zeros for AlexNet and VGG-16, respectively, and some layers in these networks have more than 90% of zeros.

This paper follows a prediction-based approach, and as the main contribution, it goes beyond zero simplifications and also considers other types of simplifications for MAC-intensive convolutions, this way increasing the chances of MAC reductions. To the best of our knowledge, this is the first work examining the likelihood of factorizing a computationally complex CONV to less complex computations in forms of aggregation functions, such as *pooling operations*. The convolutions with this property are referred to as *reducible convolutions*, and the activations with their corresponding CONV identified as reducible are called *reducible activations*, throughout this paper.

To identify reducible CONVs, we proposed a multi-class CNN-based predictor, called *reducible- and zero-activation predictor (RedZAP)*, which uses the lightweight CNN structure of a binary predictor, called ZAP [68]. Moreover, a modified version of the three-step prediction approach proposed in [68] was used for our reducible-CONV prediction. First, a subset of output activations is fully computed according to a pre-defined pattern. Second, fed with these initially computed activations, the employed predictor predicts the reducible possibility of the convolution operations related to the remaining activations and determines the type of simplification technique to be applied to those predicted as reducible. Finally, the activations marked as reducible are processed according to the determined simplification technique, skipping the original CONV operation, while other activations are fully computed.

The main contributions of our proposed reducible-CONV prediction approach is summarized as follows:

- **Going beyond zero-valued activation simplifications.** Unlike prediction-based works, such as ZAP, which merely consider zero simplifications, RedZAP pushes the horizon of CONV simplifications. In fact, RedZAP not only identifies reducible CONVs, which is analogous to the prediction of zero-valued activations in ZAP, but also determines the type of simplification technique applicable for each CONV predicted as reducible.
- **Being compatible and friendly to existing CNN architectures.** The simplification techniques employed in our approach can be simply applied using pooling operations, which are very common in CNNs.
- **Applying an adaptive and flexible pooling-based CONV factorization.** In contrast to resource-efficient model designs such as MobileNet [4], which apply the CONV factorization at the architecture level for all the activations, RedZAP determines the applicability of factorization for each individual activation based on the received input data and desired quality at a given time. In addition, unlike MobileNet, which factorizes a CONV into simpler but still CONV operations, our work considers CONV factorization with pooling operations, having lower computational costs compared to CONV operations.
- **Creating a wide range of energy-quality trade-offs.** Our approach creates a wide range of trade-offs between energy and quality, exploiting the inherent data redundancy existing in CNNs.

This paper is organized as follows: In Section 2, a detailed overview of the related work is presented. Section 3 provides preliminary knowledge on the structure of CONV layers in CNNs and the three-step zero-activation prediction approach proposed in [68]. Section 4 is devoted to the proposed convolution factorization approach by covering six topics. First, the intuitive idea behind reducible CONVs is introduced. Second, the definition of different types of reducible convolutions is presented. Third, the concept of reducible convolutions from the architectural perspective is demonstrated. Fourth, the steps for identifying reducible convolutions are discussed. Fifth, the structure of the RedZAP predictor is presented. Finally, the MAC saving and storage overhead associated with our proposed approach is discussed. Section 5 includes the experiments and presents the results and discussions, and concluding remarks are made in Section 6.

2. Related work

An *ML task*, such as image classification and object detection, is basically a set of elementary operations, i.e., MAC operations, performed toward the expected output. To give an illustration, in the task of image classification using a CNN, the label of an unknown image is determined by performing a large number of CONV operations. Each CONV can be considered as a sequence of MAC operations. With these considerations in mind, the energy associated with an ML task, denoted by E_{task}, is influenced by the number of the involved elementary operations, i.e., N_{oper}, and the energy of an individual elementary operation, namely, E_{oper}, demonstrated as follows:

$$E_{task} = N_{oper} \times E_{oper} = N_{oper} \times act \times cap \times vol^2 \times frq, \tag{1}$$

in which act, cap, vol, and frq refer to the activity coefficient, switched capacitance, voltage, and frequency, respectively.

According to (1), the literature has exploited two types of knobs to save energy: (i) E_{oper}, and (ii) N_{oper}. Some works [69–72] mitigate the dynamic energy of an elementary operation by designing approximate adders and multipliers. Further, an enormous body of research has been devoted to mitigating the high computational complexity of CNNs by reducing the number of MAC operations. Similarly, this paper exploits the number of MAC operations as leverage for energy saving. The studies targeting N_{oper} benefit from five mechanisms to reduce the number of MAC operations: (i) resource-efficient DNN model design, (ii) neural architecture search, (iii) DNN model compression, (iv) DNN hardware acceleration, and (v) zero-valued activation prediction.

Resource-efficient DNN model design. During the last decade, enormous efforts have been directed toward proposing resource-efficient DNN model designs with reduced complexity, aiming at increasing chances of their deployment on mobile and embedded devices with strict resource limitations.

GoogleNet [8] uses parallel connections to increase the depth of CONV layers, this way achieving extremely less complexity compared to the conventional designs in which CONV layers are stacked using only serial connections. ResNet [3,9] goes even deeper in CONV layers by employing special *skip connections* to address the *vanishing gradient* challenge, arising when training very deep DNNs. With this challenge, during the error back-propagation, the gradients considerably shrink when reaching the very early layers of very deep networks, affecting the weight updating in these layers. Squeeze-and-excitation networks (SENets) [10] are built by stacking novel architectural units, called *SE blocks*. These blocks can be incorporated into the existing DNN model designs to improve their performance, in terms of accuracy, at the cost of negligible computational and storage overhead. Although the computational overhead in GoogleNet and ResNet is still huge, i.e., on the order of billions of MAC operations, they incorporate more CONV layers with much less computational complexity compared to their preceding variants, such as VGG-16.

Leading DNN model designs such as GoogleNet and ResNet attempt to achieve state-of-the-art performance while keeping computational complexity low; however, they have not yet fully explored the low computational complexity opportunities, especially required for resource-constrained devices. Therefore, various works in the literature aim at enabling further MAC-saving benefits, i.e., on the order of hundreds or even tens of million MAC operations, targeting mobile and embedded devices. Examples of such works include SqueezNet [11], MobileNet, ShuffleNet [12], CondenseNet [13], ShiftNet [14], you only look once (YOLO) [15–17], and single shot multibox detector (SSD) [18].

SqueezNet achieves 50x fewer parameters and 510x smaller model size (i.e., less than 0.5MB), compared to AlexNet, by employing computationally inexpensive 1×1 convolutions and reducing the kernel sizes. MobileNet takes a CONV factorization approach and reduces a standard CONV operation into two simpler CONV operations. ShuffleNet uses the concepts of *group convolution* and *channel shuffle* to reduce MAC operations. Although the concept of group convolution significantly reduces the computational cost by partitioning the input activations in each layer into disjoint groups, with each group generating its own outputs, it decreases the chances of feature reusing across the network. CondenseNet resolves this drawback by learning group convolutions, and this way enabling effective reuse of features throughout the network, especially those features that reusing them in the subsequent layers is beneficial in terms of performance. ShiftNet substitutes spatial convolutions with parameter-less and MAC-less shift operations. YOLO and SSD are real-time single-stage object detectors that determine both the locations and class probabilities of objects in an image through a single CNN in one evaluation step, in which the detection time is much faster compared to the sequential performing of these steps. The real-time feature of YOLO comes at the expense of incurring more localization errors, as compared to the state-of-the-art object detectors.

Neural architecture search (NAS). NAS attempts to obtain a neural architecture with the best possible performance in an automated way. This automatic architecture search can be favored over handcrafted architectures in that the design process for handcrafted architectures is time-consuming and needs major expertise. Moreover, in handcrafted architectures, achieving the best possible performance by tweaking their hyper-parameters is probably achievable with huge trial-and-errors [23].

For NAS, there is a rich and complicated literature, and excellent surveys already exist on which [19–21]. However, the main point to be highlighted here is that while most of the proposed NAS techniques target solely performance, in terms of prediction accuracy, there are works [22–25] which consider other factors, including resource limitations, taking a multi-objective approach on NAS. As a result, these works are more suited for scenarios when a neural architecture is required to be deployed on a resource-constrained device.

Model compression. Some works attempt to make existing DNN model designs resource efficient by employing compression techniques, which can be classified into: (i) pruning [26–33], (ii) quantization [34–41] and binarization [42–44], and (iii) knowledge distillation [45–50]. With *pruning*, the parameters that have the least important effects on the quality of the final output, e.g., zero weights, are eliminated. *Quantization* approximates DNN parameters in floating-point format with low-bit width numbers and enables DNN computations in reduced precision; this way power-hungry floating-point multiplications are avoided. *Binarization*, as an extreme case of quantization, restricts the precision of weights to only one-bit representations, introducing *binary neural networks (BinNets)*. The idea of BinNets was first introduced in BinaryConnect [42], in which weights only take values of $\{-1, 1\}$ considering their signs. In addition to the weights, output activations can be binarized, introducing binarized neural networks (BNNs) [43]. In BNNs, CONV operations are simplified into inexpensive XNOR operations. BinnaryConnect and BNNs exhibit high degrees of compression, leading to considerable accuracy degradation. To address this issue, binary weight networks (BWNs) and XNOR networks [44] have been proposed, attempting to reduce the difference between binarized weights or activations and their original values. *Knowledge distillation*, however, takes a resource-intensive powerful DNN, and generates an approximate less complex resource-efficient DNN.

Hardware acceleration. DNN hardware accelerators based on energy-efficient technologies, such as application-specific integrated circuits (ASIC) and field-programmable gate arrays (FPGAs), have exhibited remarkable achievements in improving the resource efficiency of DNNs. DianNao [51] was the first accelerator aiming at reducing memory access, especially for large DNNs. ShiDianNao [52] is a variation of DianNao, targeting embedded devices. CORN [53] and UCNN [54] use *computation reuse* for the huge amount of inherent repetitive computations existing in DNNs. These computations, in CORN, are multiplication operations having operands with the same or very close values, and in UCNN, are dot products with the same weights repeated across kernels. Similarly, [55] employs the computation reuse for consecutive executions of a DNN, and this way, avoids an entire DNN execution in case of receiving similar inputs. This can be beneficial in applications such as speech recognition and video classification, which involve multiple executions of a DNN to process a sequence of inputs (e.g., audio frames and images). PRA [56] avoids ineffectual computations, in terms of zero-valued product terms, within a multiplication operation involved in a DNN. ΔNN [57] enables computation reuse for the calculations involved within a kernel. To achieve this, the kernel weights are sorted in ascending order, with each weight stored as its difference (Δ) to the previous value. Thus, the computation associated with each weight is performed by reusing the computation of the nearest weight followed by a calculation on the Δ value. The authors quantize the Δ values to further reduce the computation complexity.

Zero-valued activation prediction. Another line of works directed towards improving the energy efficiency of DNNs is to predict computations with tolerable or no effect on the application's output quality, and next, to skip or mitigate their computational complexity. Among such computations are those incurred by zero-valued activations. Therefore, zero-prediction, or sign-prediction, has been the focus of many studies, and it usually requires an initial computational overhead to decide whether to perform the computations pertaining to the remaining part.

The works [58–60] utilize the result from the partial processing related to the MSB part of a convolution operation to predict the sign of an activation, hence deciding on whether performing or skipping the processing related to the LSB part. Some works target the operations involved within a single convolution. As an example, in [61] the sequence of MACs required to generate a single activation is reordered such that the results from the very first MAC operations lead to the early prediction of the activation's sign, skipping the remaining MAC operations in the sequence. Other works make use of a predictor to

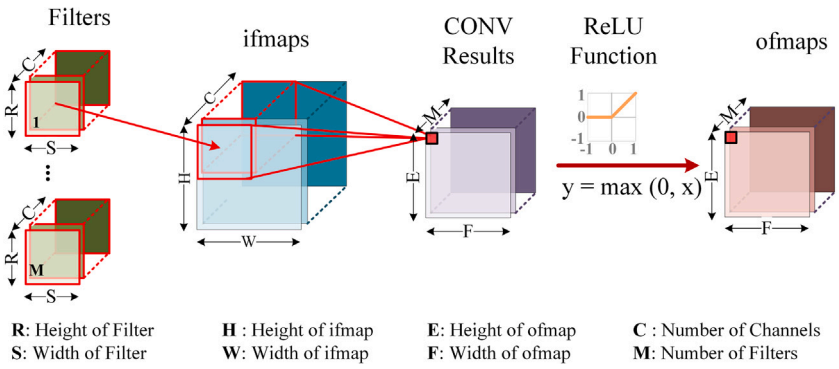


Fig. 1. Illustration of a CONV layer in CNNs [73].

identify zero activations. As an example, the work in [68] uses a lightweight CNN-based predictor to identify zero activations in a CONV layer, and [62] employs a *fully-connected layer(FC)* layer to predict the sign of each activation. [64] applies the concept of zero-activation prediction to the structure of a CNN by incorporating low-cost collaborative layers, which act as feature masks for the original CONV layers. Some works [63,65,67] consider the spatial correlation characteristic of CNNs as the basis for their prediction. By spatial correlation, it is meant that adjacent output activations share close values. These works make use of this characteristic for zero-prediction [63,67] or interpolation of the missing values [65]. The work proposed in [66] considers the prediction of activations highly close to zero, given a threshold, in addition to zero activations.

Our work is not focused on proposing new resource-efficient model designs for DNNs, and instead, it targets existing DNN models and attempts to improve their energy efficiency. However, this work is similar to works like MobileNet in that it presents a convolution factorization approach, with a clear distinction that the factorization is applied to each individual activation. Moreover, unlike the works that exploit compression techniques to eliminate redundant computations prior to the inference phase, and this way, considerably reducing the model size, this work skips or simplifies these computations at the inference phase without applying any permanent elimination. In fact, in this work, the set of activations whose computations are skipped or simplified varies during the inference phase, based on the input data fed into the DNN and the desired quality. To identify these redundant computations, this work makes use of a light-weight CNN-based predictor similar to ZAP, and this makes our work very closely related to the works taking a prediction approach. The main difference is that in addition to predicting zero and near-zero activations, which is very common in the literature, this work attempts to consider other forms of simplifications, although no hardware acceleration support is presented.

3. Background

This section provides preliminary knowledge on (i) CONV layers, which drive the most part of high computational demands of CNNs, and (ii) ZAP, which acts as the basis for our proposed reducible-convolution prediction approach.

3.1. Convolutional layers in CNNs

A CNN is composed of several CONV layers followed by a few FC layers. Each CONV layer extracts successively higher-level abstractions of the input data, called *feature maps (fmaps)*, for use in the FC layers in order to determine the *classification* output. However, the CONV layers are involved with a huge amount of MAC computations, taking the majority portion (i.e., more than 90% [5]) of total MAC operations in a CNN, as compared to FC layers.

Fig. 1 illustrates the computation within a CONV layer. Within each CONV layer, a number of 3-D arrays of weights, i.e., kernels, are applied to the layer's input activations. The set of input activations fed into a CONV layer is a 3-D array of *input features*, which is measured in the preceding CONV layer and is known as a single 3-D *input feature map (ifmap)*. By sliding all kernels over all subareas, also called *receptive fields*, within the ifmap and performing the convolution operation, a 3-D array of *output features* is generated, referred to as a single 3-D *output feature map (ofmap)*.

The computation associated with a single output activation $X_{out}[z][y][x]$, belonging to the 3-D ofmap X_{out}, is summarized as [73]:

$$X_{out}[z][y][x] = B[z] + \sum_{l=0}^{C-1}\sum_{j=0}^{R-1}\sum_{i=0}^{S-1} X_{in}[l][Uy+j][Ux+i] \times Kernel_z[l][j][i],$$
$$0 \le z \le M, 0 \le y \le E, 0 \le x \le F,$$
$$E = \frac{H-R+U}{U}, F = \frac{W-S+U}{U}, \tag{2}$$

in which $x_{in,[C\times H\times W]}$ represents the ifmap array with C known as the number of *input channels*, and W and H representing the width and height of each input channel, respectively; $Kernel_{i,[C\times R\times S]}$ is the ith *kernel*, in which there is a 2-D array of weights corresponding to each input channel, with S and R representing width and height of these arrays; M determines the number of kernels and also the number of *output channels*, and F and E represent the width and height of each output channel in the ofmap, respectively; B is the biases vector; and U is the stride size.

The number of MAC operations and parameters within a CONV layer are respectively calculated as (3) and (4):

$$Op = (M \times E \times F) \times Op_{act}, \tag{3}$$

$$Par = M \times (C \times R \times S), \tag{4}$$

wherein Op_{act} is defined as the number of MAC operations required to compute a single output activation and is calculated as $C \times R \times S$.

Moreover, in CNNs, to reduce an ofmap's dimensions, an average or max aggregation function is applied to each receptive field, a process referred to as *average* or *max pooling*, respectively. The processing associated with these functions involves a sequence of add operations. Intuitively, considering a multiply operation computationally more expensive than an add operation, the computational overhead of a pooling function is approximately defined as:

$$Op_{act}^{pool} = \frac{R \times S}{\gamma}, \gamma > 1, \tag{5}$$

in which γ is an integer value determining the relative computational complexity of multiply and add operations. According to [74], in this paper, γ is assumed to be three, implying that a multiply operation is equivalent to three add operations, from the computational complexity perspective.

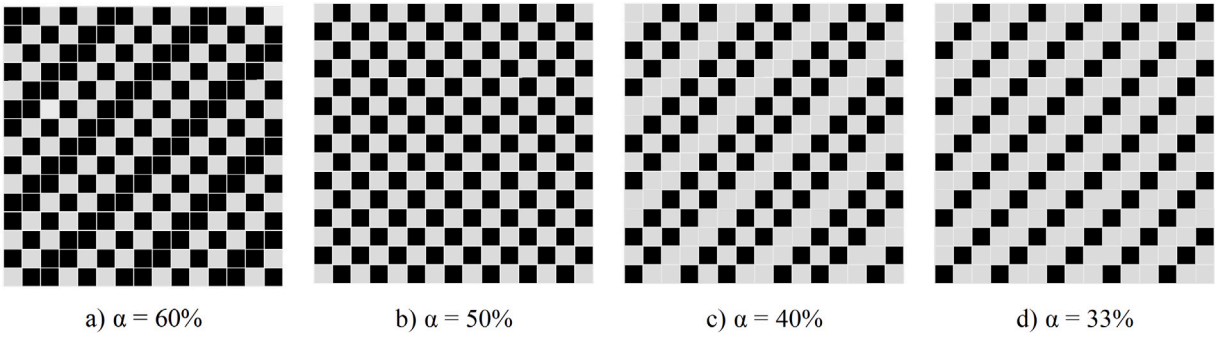


Fig. 2. Illustration of different types of computation masks, i.e., (a) mask *A*, (b) mask *B*, (c) mask *C*, (d) mask *D*, differing in α, defined as the ratio of the number of initially computed activations to the ofmap size. Black squares represent the set of activations initially computed using a standard CONV operation [68].

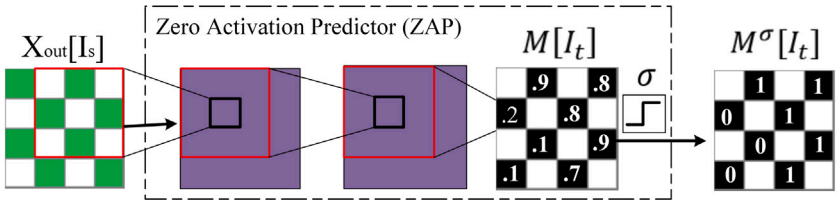


Fig. 3. General architecture of ZAP, proposed in [68].

3.2. The three-step zero-activation prediction approach

The authors in [68] make use of a ZAP predictor to identify and skip zero-valued output activations within a CONV layer. To predict these activations, an initial computational overhead is required. This initial computation involves performing complete standard CONV operations for an initial set of activations, denoted by I_s. Next, fed with this partially computed ofmap, denoted by $X_{out}[I_s]$, the ZAP predictor determines the potential zero-valued activations among the remaining activations, i.e., I_t.

The set of initially computed activations is determined using a number of pre-defined *computation masks*, shown in Fig. 2, in which the black and white squares represent the I_s and I_t sets, respectively. The masks differ in a parameter called α, defined as the ratio between the cardinality of set I_s and the ofmap's dimensions. The masks enable different levels of trade-offs between MAC savings and quality in that larger values of α increase the accuracy of prediction, whereas reducing the number of skipped activations.

The three-step zero-prediction approach is demonstrated in what follows.

- **Step 1.** Given a mask type, $X_{out}[I_s]$ is calculated.
- **Step 2.** Fed with $X_{out}[I_s]$, the ZAP predictor generates a mask, denoted by $M[I_t]$, containing the prediction scores for the unprocessed remaining activations (i.e., I_t). In order to transform these scores into the binary representation, the authors of [68] applied a threshold σ to $M[I_t]$, generating $M^{\sigma}[I_t]$ as the predictor's output.
- **Step 3.** According to $M^{\sigma}[I_t]$, the activations in I_t predicted as zero are skipped, while the others are fully computed similar to the activations in I_s.

In [68], a two-layer CNN is used for the ZAP predictor, as shown in Fig. 3. To mitigate the overhead associated with the employed CNN model, the authors have benefited from *depthwise convolution (DW-CONV)* layers proposed in MobileNet. DW-CONV layers use only 2-D kernels with the channel's depth eliminated. This elimination makes each standard CONV operation, with $C \times R \times S$ MAC operations, factorized into a DW-CONV, with $R \times S$ MAC operations, followed by a 1×1 convolution known as a *pointwise convolution (PW-CONV)*, composed of C MAC operations.

The authors in [68] showed that the MAC reduction of a DW-CONV compared to a standard CONV operation can be formulated as $1/C$. Since the number of channels in a CNN is usually greater than 100 [3,6,7], the computational overhead of ZAP is concluded to be negligible. The authors [68] presented a similar declaration and concluded the storage overhead was negligible, as well.

4. The proposed convolution factorization approach

This section discusses the reducible-convolution prediction approach, proposed in this paper, by covering six topics. First, the motivation and intuitive idea behind reducible CONVs are described. Second, the precise definition of reducible CONVs and their different types are presented. Third, the architectural view of reducible convolutions is presented. Fourth, the three-step approach for identifying reducible CONVs is demonstrated. Fifth, the structure of the proposed RedZAP predictor is clarified. Finally, a thorough analysis of associated MAC saving and storage overhead is presented.

4.1. The intuitive idea behind reducible convolutions

The intuitive idea behind this paper is to simplify the computation associated with a MAC-intensive CONV whenever applicable. To clarify, consider (2) with $C = 1$, showing the computation required per input channel for calculating a partial value of an output activation. This computation is a weighted sum of input activations and the kernel weights, and by considering a kernel size of k (i.e., assuming $R = S = k$ in (2)), it involves k^2 MAC operations. The objective is to obtain an approximate value for the activation by simplifying this computation such that the number of required MAC operations decreases below k^2.

One way of simplifying a CONV is to factor either (i) the X_{in} variable or (ii) the $Kernel_z$ variable out of the weighted sum shown in (2). An aggregated value, obtained by applying max and average aggregation functions, for example, can be used for the extracted variable. Factorizing a CONV based on either variable using max and ave aggregation functions will result in a sum aggregation for the other variable. This way, the CONV computation reduces into the product of

two computationally low-cost aggregation functions, i.e., (i) *the input aggregation function* $F(.)$, applied to the receptive field, and (ii) the *kernel aggregation function* $G(.)$, applied to the kernel array.

The aggregation functions $F(.)$ and $G(.)$ must be selected such that the computational complexity of a factorized CONV decreases below k^2 MAC operations. In our previous work [75], it was shown that since the kernel aggregation function can be applied to the kernel weights prior to the inference phase, only the computational overhead involved with the input aggregation function contributes to the computational complexity of the factorized CONV. Moreover, factorizing a CONV is accepted only if it leads to a tolerable error in the application's final output. In this paper, this is controlled using a threshold, called a *simplification threshold*. In our CONV factorization approach, only activations that the difference between real and factorized values is below a given simplification threshold are considered reducible.

With all these considerations in mind, an activation is called reducible if the corresponding CONV operation can be factorized into the product of two aggregation functions, i.e., $F(.)$ and $G(.)$, attributed to two conditions: (i) the difference between the real and factorized values is below a given simplification threshold, and (ii) the processing associated with $F(.)$ involves extremely less than k^2 MAC operations. In this paper, the terms reducible convolution and reducible activation refer to the same concept and are used interchangeably.

4.2. The definition of reducible convolutions

The definition of a reducible activation is presented as follows.

Definition 1 (*Reducible Activation*)**.** Given the input and kernel aggregation functions $F(.)$ and $G(.)$, respectively, and the simplification threshold ϵ, an output activation $X_{out}[z][y][x]$ is *reducible* if the following two conditions are met: (i) the computational complexity of $F(.)$ for a single input channel, denoted by $Op_{F(.)}$, satisfies:

$$Op_{F(.)} \leq \frac{k^2}{\beta},\ \beta > 1, \tag{6}$$

in which k is the kernel size and β is the maximum integer value greater than one satisfying (6), and (ii) for an approximate value $\hat{X}_{out}[z][y][x]$ calculated using:

$$\hat{X}_{out}[z][y][x] = B[z] + \sum_{l=0}^{C-1} F(X_{in}[l][Uy : Uy+k][Ux : Ux+k]) \times G(Kernel_z[l]) \tag{7}$$

condition (8) holds:

$$\Delta = Norm_{[0,1]}(|X_{out}[z][y][x] - \hat{X}_{out}[z][y][x]|) \leq \epsilon,\ 0 \leq \Delta \leq 1,\ 0 \leq \epsilon \leq 1, \tag{8}$$

in which Δ is the difference value between the activation's value and its approximated value, normalized between zero and one in each CONV layer using the normalization function $Norm_{[0,1]}(.)$.

Selecting aggregation functions $F(.)$ and $G(.)$ for our CONV factorization is influenced by the fact that the factorization is whether based on the input activation variable or based on the weight variable. In this paper, the factorization is based on the input activation variable, as demonstrated in our previous study [75].

For the input aggregation function, max- and ave-pooling operations were used in this paper, thus making the kernel aggregation function equal to a PW-CONV. Although it is not necessarily needed for $F(.)$ to be restricted to max and ave pooling functions, the rationale behind selecting these functions is two-fold. First, these functions are supported by the excising CNN architectures as they are the most widely used pooling operations in CNNs and exhibit appealing properties discussed in [76]. Second, according to (5), the computational complexity of these pooling operations (i.e., $k^2/3, \beta = 3$) is considerably less than the computational overhead of a CONV operation, and this satisfies (6).

In this paper, three types of reducible activations are defined: (i) max reducible, (ii) ave reducible, and (iii) zero reducible. *Max-* and *ave-reducible* activations have max- and ave-pooling operations as their input aggregation functions, respectively. Definition 2 clearly defines these types of activations. To further increase the chances of MAC reductions, this paper also considers zero and near-zero activations, referred to as *zero-reducible* activations.

Definition 2 (*Max- and Ave-Reducible Activations*)**.** Given a simplification threshold ϵ, an output activation $X_{out}[z][y][x]$ is *max reducible* or *ave reducible* if for an approximate value $\hat{X}_{out}[z][y][x]$ respectively calculated using (9) or (10):

$$\hat{X}_{out}[z][y][x] = B[z] + \sum_{l=0}^{C-1} \max_{\substack{Uy \leq j \leq Uy+k, \\ Ux \leq i \leq Ux+k}} \{X_{in}[l][j][i]\} \times \sum_{i=1}^{k^2} Kernel_z[l][i], \tag{9}$$

$$\hat{X}_{out}[z][y][x] = B[z] + \sum_{l=0}^{C-1} \underset{\substack{Uy \leq j \leq Uy+k, \\ Ux \leq i \leq Ux+k}}{ave} \{X_{in}[l][j][i]\} \times \sum_{i=1}^{k^2} Kernel_z[l][i], \tag{10}$$

condition (8) holds.

4.3. The architectural view of reducible convolutions

From the architectural perspective, Definition 2 is illustrated in Fig. 4. As shown, a CONV operation with a kernel size of $R \times S$, associated with an output activation, is replaced with two operations: (i) a max or average pooling of size $R \times S$, applied to each input channel of the corresponding receptive field from the ifmap, and (ii) a PW-CONV, invoked across the input channels between the aggregated input features and their corresponding aggregated kernel weights. Unlike the PW-CONV, the pooling operation is shared between all the output activations belonging to different output channels but having the same width indices and the same height indices.

This architectural view holds only when two conditions are met: (i) each output activation in the ofmap is either max or ave reducible, and (ii) the activations with the same width indices and the same height indices are either all max reducible or all ave reducible. However, in practice, this is not the case, and thus, the factorization is performed only for CONVs identified as reducible.

4.4. The steps for identifying reducible convolutions

To improve the energy efficiency of an existing CNN model, called a *host CNN*, RedZAP predictors are incorporated into the desired CONV layers, and using a three-step method, reducible convolutions in each layer are identified and factorized. The employed three-step method is a modified version of the zero prediction approach used in ZAP, mainly differing in steps two and three. The steps, illustrated in Fig. 5, are as follows:

- **Step 1.** The partial ofmap $X_{out}[I_s]$ is generated, in a similar way taken in ZAP.
- **Step 2.** Similar to ZAP, receiving $X_{out}[I_s]$ as the input, the predictor generates a computation mask for the remaining activations. However, unlike the binary mask in ZAP, composed of only zeros and ones representing no computation and complete computation, respectively, the mask from RedZAP contains a range of integer values, each value corresponding to a specific type of a reducible CONV.
- **Step 3.** Similar to ZAP, appropriate computation is invoked for each remaining activation, based on the mask obtained from the previous step. However, in RedZAP, a more diverse set of actions is pursued for each activation: (i) zero-reducible activations are skipped similar to ZAP, (ii) max- and ave-reducible activations are

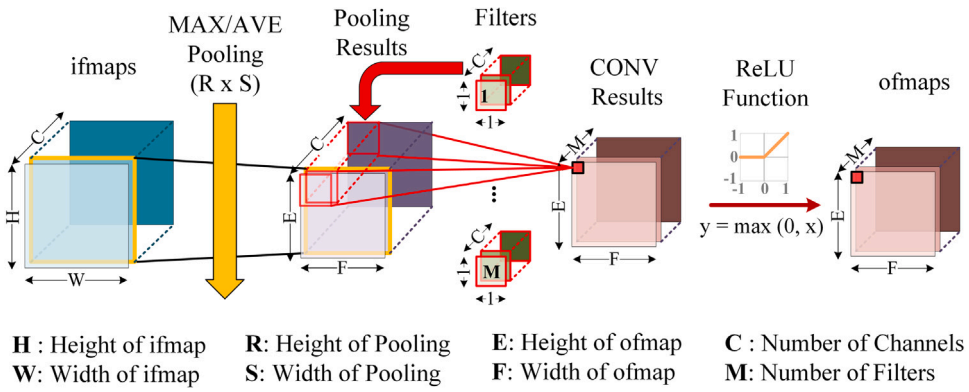


Fig. 4. Illustration of the proposed convolution factorization approach from the architectural perspective.

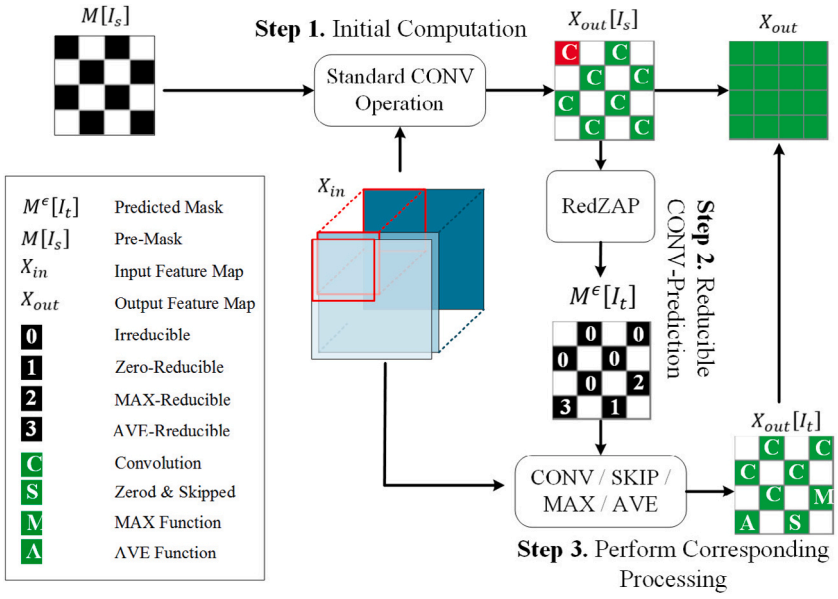


Fig. 5. Illustration of the steps required to identify reducible activations, as a modified version of the three-step zero-prediction method proposed in [68].

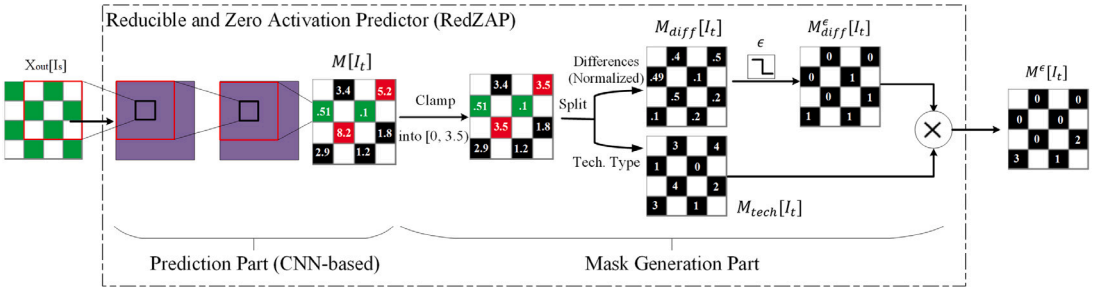


Fig. 6. Illustration of the structure of the proposed RedZAP predictor.

factorized using (9) and (10), respectively, and (iii) irreducible activations are fully computed using a standard CONV operation according to (2).

4.5. The structure of RedZAP predictor

The structure of RedZAP, illustrated in Fig. 6, comprises two parts: (i) the prediction part and (ii) the mask generation part. The *prediction part* benefits from the CNN structure employed in ZAP to predict a mask for the remaining activations, denoted by $M[I_t]$. However, the predicted mask values within $M[I_t]$ are not suitable for our multi-class prediction purpose in their initial raw formats. Therefore, in the *mask generation part*, we devised a set of processing steps to transform this initially predicted mask, i.e., $M[I_t]$, into our desired multi-class mask, denoted by $M^{\epsilon}[I_t]$.

The mask generation part. Using the generated mask $M^{\epsilon}[I_t]$, a RedZAP predictor is expected to achieve two objectives: (i) discriminating reducible activations from the irreducible ones and (ii) determining the type of activations identified as reducible. To achieve these objectives, the CNN within the prediction part of a RedZAP is trained in a way that the integer and decimal parts of a predicted value within $M[I_t]$

respectively incorporate two types of information: (i) the simplification type for the corresponding activation and (ii) the normalized difference magnitude incurred if the activation is simplified using the specified simplification type. Therefore, given a simplification threshold, a reducible activation is determined using the decimal part, and if reducible, the type is specified using the integer part.

The representations for integer and decimal parts should be selected under computational overhead and predictor accuracy considerations. In this study, similar to our previous study [75], the integer values 0, 1, 2, and 3 were selected as the class labels for our RedZAP predictor, representing irreducible, zero-reducible, max-reducible, and ave-reducible activations, respectively. However, the main distinction resides in the decimal part representation. In the previous study [75], for a predicted mask value, the decimal part is extracted as a real value in the range of [0, 1). The main drawback with this representation is that during the training, the predictor ignores reducible activations that are approaching the target class label from the smaller values, decreasing the accuracy of the predictor.

To clarify, consider the simplification threshold value of 0.4. For a max-reducible activation with the predicted mask value of 2.1, the class label and the normalized difference magnitude are extracted as 2 and 0.1, respectively, and after applying the threshold, the activation is correctly identified as a max-reducible activation. However, for a max-reducible activation with the predicted mask value of 1.9, the normalized difference magnitude is extracted as 0.9, and after applying the threshold, it is identified as irreducible. This way, max-reducible activations are learned as irreducible activations.

In the current study, the difference magnitudes are normalized in the $[0, 0.5)$ range, instead of $[0, 1)$. With this selection, $[0, 3.5)$ will be the valid range for the mask values generated from the prediction part, and for a mask value m within this range, the class label and the normalized difference value are calculated as $round(m)$ and $\mid m - round(m) \mid$, respectively, in which $round(.)$ is a rounding function.

With these considerations in mind, the process involved within the mask generation part can be conceptually summarized in four steps, as follows:

First, since the prediction part may generate mask values exceeding the valid range, these mask values, highlighted with red squares in Fig. 6, are clamped to the least upper bound of the valid range (i.e., here, 3.5). This makes the corresponding activations to be predicted as irreducible, as they have received the maximum simplification difference magnitude as a result of the clamping. Next, the integer and decimal parts, represented by $M_{tech}[I_t]$ and $M_{diff}[I_t]$, respectively, are extracted. Then, the simplification threshold is applied to $M_{diff}[I_t]$, resulting in the binary mask $M^{\epsilon}_{diff}[I_t]$, composed of zeros and ones representing potential irreducible and reducible activations, respectively. For the training and inference phases, the thresholds can be different and are denoted by ϵ_{train} and ϵ_{inf}, respectively. Finally, the class labels are determined for each activation and are included within the $M^{\epsilon}[I_t]$. For activations identified as irreducible in the $M^{\epsilon}_{diff}[I_t]$ mask, the class label is zero, and for the reducible activations within $M^{\epsilon}_{diff}[I_t]$, the class labels are selected from the $M_{tech}[I_t]$ mask, accordingly. In other words, $M^{\epsilon}[I_t]$ is conceptually obtained by the dot product of $M^{\epsilon}_{diff}[I_t]$ and $M_{tech}[I_t]$.

Training. The training procedure for a single RedZAP, incorporated into a CONV layer, is illustrated in Fig. 7. During the training phase, for a mask generated by the RedZAP, denoted by $M^{\epsilon_{train}}_{pred}$, there should be a corresponding target mask, denoted by $M^{\epsilon_{train}}_{tar}$. To generate the target mask three steps are followed. First, for each output activation within the CONV layer, it is determined which simplification technique (i.e., zeroing, max pooling, or ave pooling) gives the minimum difference magnitude when applied to the activation. The simplification technique types and associated difference values, normalized in the range of $[0, 0.5)$, are stored in $M_{tech}[I_t]$ and $M_{diff}[I_t]$, respectively. Next, the *training simplification threshold* is applied to the $M_{diff}[I_t]$, resulting in the $M^{\epsilon_{train}}_{diff}[I_t]$ mask. Finally, the target mask is obtained by the dot product of $M^{\epsilon_{train}}_{diff}[I_t]$ and $M_{tech}[I_t]$.

4.6. MAC saving and storage overhead analysis

MAC saving. For the architectural view shown in Fig. 4, the computational overhead is calculated as:

$$Op^{factorizedCNN} = (C \times E \times F) \times Op^{pool}_{act} + (M \times E \times F) \times Op^{PW-CONV}_{act}. \tag{11}$$

In our previous study [75] it was demonstrated that how this architectural CONV factorization can lead to further MAC reduction opportunities, as compared to handcrafted CNN architectures such as MobileNet.

However, (11) holds only when the two conditions mentioned in Section 4.3 are met, which is an unrealistic assumption. The first condition is violated when some of the activations are predicted as zero reducible or irreducible, and thus, are skipped or computed using standard CONV operation, respectively. The second condition is violated when at least one max-reducible and one ave-reducible activation having the same width indices and the same height indices exist in two different output channels. In this case, instead of one pooling operation, two pooling operations are required: one ave-pooling operation for ave-reducible activations and one max-pooling operation for max-reducible activations. Therefore, the general form of calculating the computational overhead involved within a single CONV layer is as follows:

$$\begin{aligned} Op^{factorized\ CNN} &= (No^{max}_{pool} + No^{ave}_{pool}) \times C \times Op^{pool}_{act} \\ &\quad + (No^{max}_{act} + No^{ave}_{act}) \times Op^{PW-CONV}_{act} + No^{irr}_{act} \times Op_{act}, \\ No^{irr}_{act} &+ No^{max}_{act} + No^{ave}_{act} \le M \times E \times F, \\ No^{max}_{pool} &\le E \times F, No^{ave}_{pool} \le E \times F, \end{aligned} \tag{12}$$

in which No^{max}_{act} and No^{ave}_{act} represent the number of max- and ave-reducible activations in the ofmap, respectively; No^{irr}_{act} is the number of irreducible activations in the ofmap; and No^{max}_{pool} and No^{ave}_{pool} determine the number of required shared max- and ave-pooling operations, respectively, in which No^{max}_{pool} is calculated by counting all max-reducible activations with the same width indices and the same height indices in the ofmap as one max-reducible activation, and the same consideration holds for calculating No^{ave}_{pool}.

Storage overhead. With our proposed CONV factorization approach, aggregated values of the pre-trained kernels are required to be stored in addition to the original kernels; therefore, the number of weights is calculated as:

$$Par^{factorized\ CNN} = Par + tech_{count} \times M \times C, \tag{13}$$

in which $tech_{count}$ is the number of employed simplification techniques (in this paper, two simplification techniques, i.e., max and average pooling). As a result, the incurred storage overhead compared to (4) is calculated as:

$$\frac{Par^{factorized\ CNN}}{Par} - 1 = tech_{count} \times \frac{1}{R \times S}, \tag{14}$$

considered as a limiting factor for the number of simplification techniques that can be employed.

Considering the most common filter size of three and, also, the fact that only less than 10% of the weights in a CNN belong to CONV layers, (14) introduces approximately 2% of storage overhead involved with our proposed CONV-factorization approach.

5. Experiments

In this section, first, the model structure and the setup used for the experiments are demonstrated. Next, three series of experiments

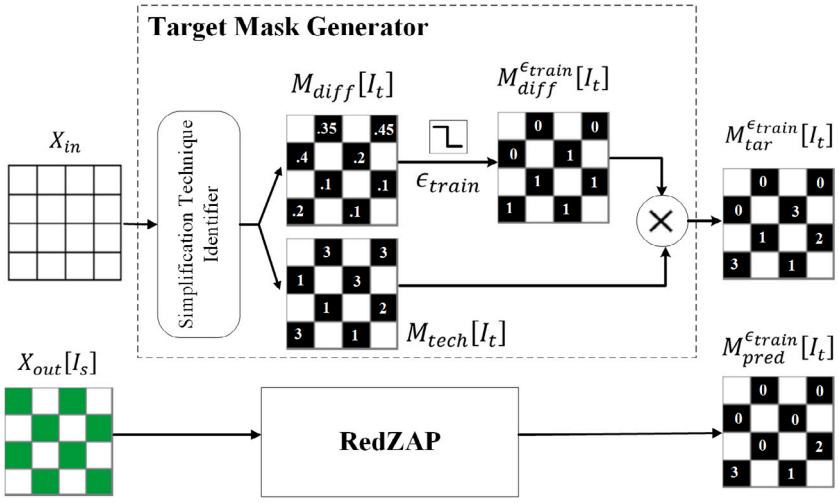


Fig. 7. Illustration of the training procedure for RedZAP.

are included, aiming at: (i) demonstrating the potential MAC reduction benefits of reducible convolutions inherent in CNNs; (ii) evaluating the performance of the proposed convolution factorization approach by comparing it with ZAP, which outperforms the other similar previous works in the literature (see [68] for details); and (iii) showing accuracy-MAC saving trade-offs offered by our trained RedZAP predictor.

5.1. Model structure and experiments setup

Model structure. The model structure used for the evaluation experiments comprises a host CNN and several RedZAP predictors, incorporated into the desired layers of the host CNN to identify reducible convolutions. The host CNN can be any CNN architecture that is targeted for MAC reductions, this way improving its energy efficiency. In this paper, a version of AlexNet pre-trained on the CIFAR-100 dataset [77], taken from [68], was exploited for the experiments. The RedZAP predictors were incorporated into each layer of AlexNet, except the first layer.

Experiments setup. To train the incorporated RedZAP predictors, the general training approach presented in [68] was followed. With this approach, each RedZAP is trained in isolation, by bypassing all the other predictors in the host CNN and forwarding the training data instances directly to the RedZAP under training. For each RedZAP, the number of epochs was set to 5 epochs, Adam [78] optimizer was used, and mean square error (MSE) was considered as the cost function.

After the training phase, the performance of RedZAP predictors was evaluated in terms of the amounts of accuracy-MAC saving trade-off they offer when incorporated into the host CNN. Additionally, to support the intuition behind the proposed convolution factorization approach, the maximum amount of MAC-saving benefits that can be offered by reducible convolutions was extracted.

Although, in our implementations, the difference magnitudes are normalized in the [0, 0.5) range, the training and inference simplification thresholds are reported in the range of [0, 1) in order to align our results with the previous studies in this context. Two simplification threshold values of 0.01 and 0.1 were used for the training phase, and for the inference phase, two types of threshold ranges were considered as a grid search to observe the behavior of RedZAP predictors: (i) a coarse-grained threshold range between 0.0 and 0.5 in steps of 0.1 and (ii) a fine-grained threshold range between 0.0 and 0.1 in steps of 0.02 to observe the behavior of RedZAP predictors for threshold values close to 0. Moreover, MAC savings are calculated using (12).

5.2. Potential reducible convolutions inherent in CNNs

Fig. 8 shows the further MAC reduction opportunities that are inherently provided by max- and ave-reducible activations, as compared to the MAC savings offered by the conventional zero-activation skipping. Additionally, near-zero activation simplifications were considered to further increase the amount of MAC reductions. These intrinsic MAC reductions using reducible convolutions are reported per simplification threshold and per CONV layer, i.e., averaged over all the thresholds, in Fig. 8(a) and Fig. 8(b), respectively. In what follows, the observations and corresponding discussions are presented.

Observation 1. The number of CONV reductions increases with the simplification threshold value, although the number of zero-valued activations remains intact, regardless of the threshold value. Moreover, with the simplification threshold set to 0.0, MAC reductions were provided only by zero-valued activation simplifications.

This observation supports the idea of considering a non-zero simplification threshold, presented in Definitions 1 and 2, in order to benefit from MAC savings using max- and ave-reducible activations, as well as near-zero activations.

Observation 2. On average, near 80% MAC reduction chances using reducible convolutions were observed, among which almost 9% pertained to MAC reductions using max- and ave-reducible activations, and near 5% pertained to near-zero activation simplifications.

Observation 3. Among max- and ave-reducible activations, ave-reducible activations contributed to more MAC-saving opportunities.

This observation highlights the effect of the selected aggregation function type on the amount of MAC reduction opportunities that can be extracted. Here, the average aggregation function is probably a more appropriate candidate for simplification. This is because the average aggregation function applied to a receptive field is more representative compared to the max aggregation function.

Observation 4. By going deeper through layers of a CNN, the number of max- and ave-reducible activations as well as near-zero activations reduces.

This Observation suggests that approximation using reducible activations is more applicable in the very first layers of a CNN. This can be intuitively explained by the fact that the feature values of the very early layers, very close to the input data, inherit higher degrees of inherent redundancy residing in the input data, which makes them more correlated. As a result, aggregation functions such as max and

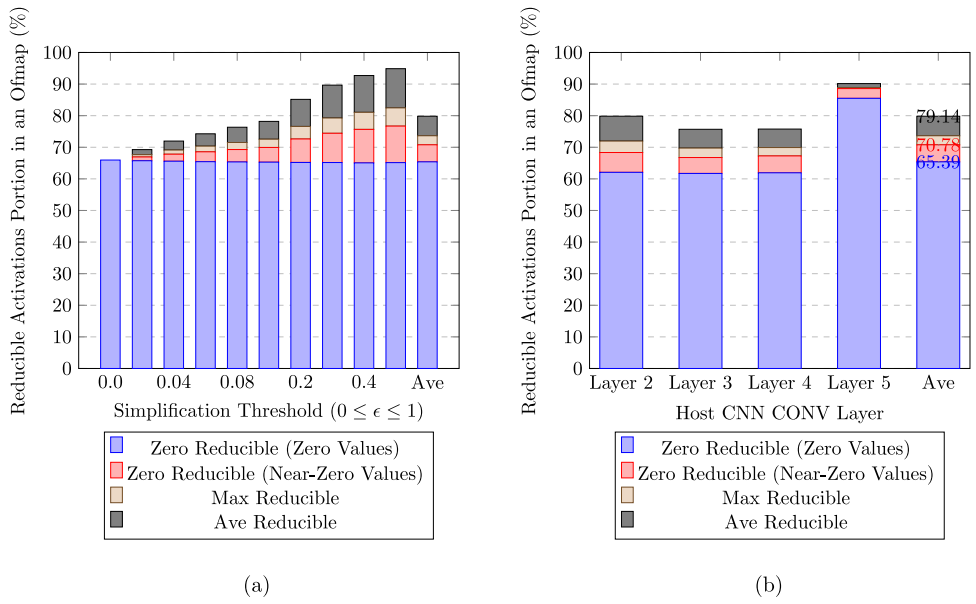


Fig. 8. Shares of different types of reducible convolutions in an ofmap for (a) each simplification threshold and (b) each layer in the host CNN.

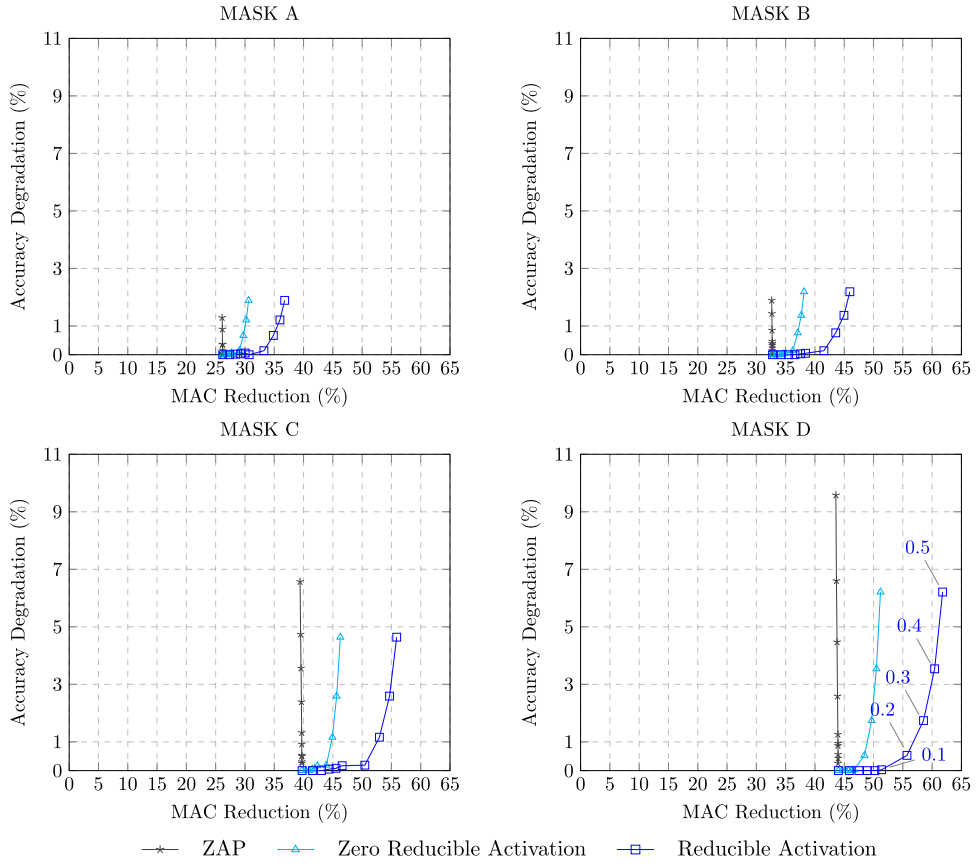


Fig. 9. Presenting MAC saving and top-1 accuracy loss trade-offs for different mask types and different inference thresholds, considering ideal ZAP and RedZAP predictors. As a sample, the "Mask D" plot shows the order of measurements corresponding to inference thresholds for the other plots. As shown, for inference thresholds pertaining to the fine-grained range, the accuracy losses are approximately zero.

average can most benefit from this correlation. The correlation between feature values decreases as layers get deeper, leading to a huge amount of errors by applying max- or average-pooling operations, even much more than zeroing the activations.

5.3. Performance evaluation of the proposed convolution factorization approach

Fig. 9 shows the trade-off between MAC saving and accuracy considering ideal ZAP and RedZAP predictors, i.e., trained with 100%

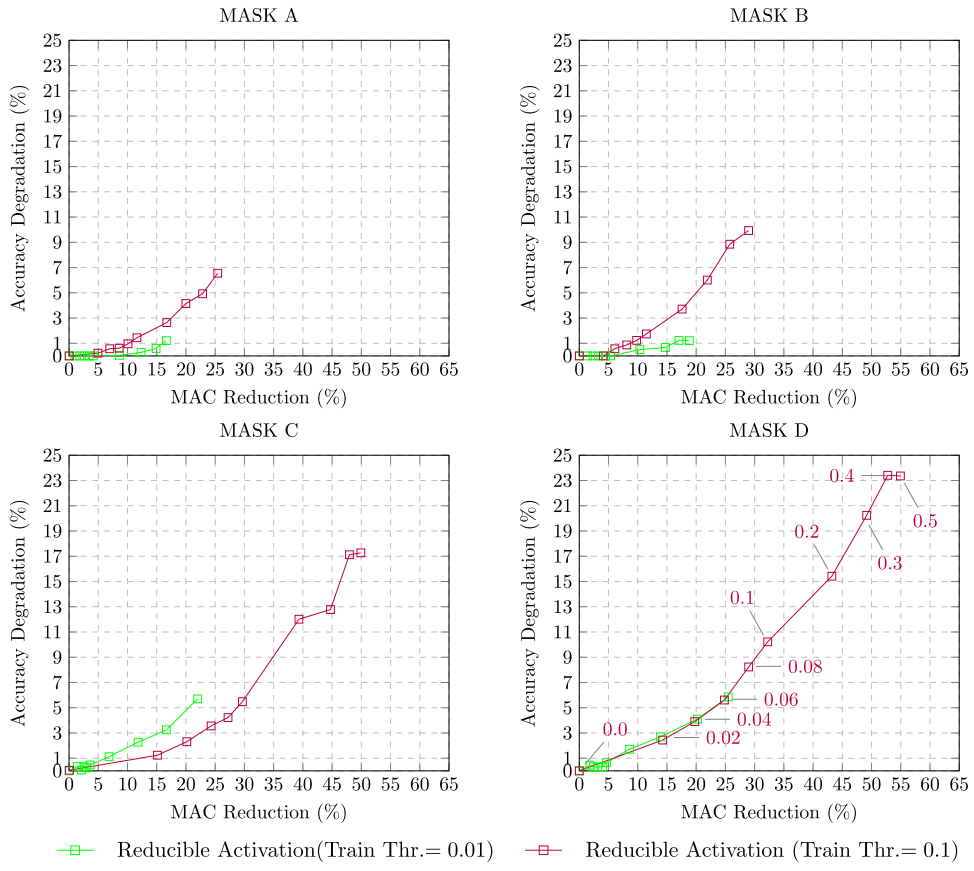


Fig. 10. Presenting MAC saving and top-1 accuracy loss trade-offs for different mask types and different training and inference thresholds, using trained RedZAP predictors. As a sample, the "Mask D" plot shows the order of measurements corresponding to inference thresholds for the other plots.

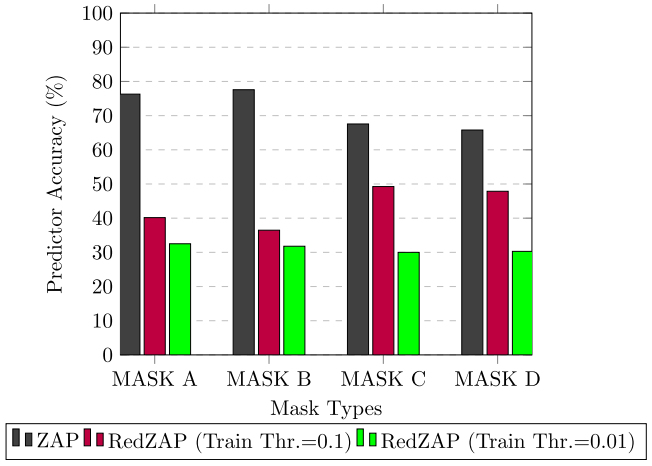


Fig. 11. Comparing accuracy of RedZAP and ZAP predictors for different mask types. The accuracy value of a RedZAP trained with a specific training threshold is averaged over the inference thresholds.

accuracy. This way the effect of predictors' accuracy on the results is eliminated, thus enabling the evaluation of the idea behind reducible convolutions. The trade-offs cover different mask types and different inference thresholds. Moreover, the trade-offs are also provided for zero-reducible activations, which are analogous to zero-activation prediction in ZAP. The following observations are made.

Observation 5. For all the mask types, the reducible activation charts were below the zero-reducible charts.

This observation supports our claim in that given a specific accuracy loss constraint, there are further MAC reduction opportunities using convolution factorizations, i.e., max- and ave-reducible activations, in addition to the MAC savings offered by zero-reducible activations.

Observation 6. For all the mask types, zero-reducible charts were below zero-activation prediction offered by ZAP.

This observation implies that even the amount of MAC reductions using zero-reducible activations are more than zero-activation predictions using ZAP. This is because zero-reducible activations include near-zero activations in addition to zero activations.

Observation 7. For all the mask types, the reducible activation charts were below the ZAP charts.

This observation supports our main claim in that given a specific accuracy loss constraint, there are MAC-saving opportunities using types of simplifications other than the conventional zero-activation simplifications.

Observation 8. With ideal ZAP and RedZAP predictors and considering the 1% top-1 accuracy loss as the target accuracy constraint, near 56% MAC reductions were achieved with our convolution factorization approach, among which 8% pertained to max- and ave-reducible activation simplifications. This MAC-saving benefit was achieved by selecting mask type D and inference simplification threshold value of 0.2. However, in the case of ZAP, the maximum MAC saving under the same accuracy loss constraint was near 44% by selecting mask type D and inference simplification threshold value of 0.02.

This observation seems to highlight the superiority of our proposed convolution factorization idea over the conventional zero-activation simplifications which are common in the literature, in terms of maximum MAC savings offered under some specified target accuracy loss

constraint. Moreover, with 56% MAC reductions in a resource-intensive CNN such as AlexNet with 600M MAC operations, our CONV factorization approach seems analogous to handcrafted architectures such as MobileNet and ShuffleNet, which target mobile and embedded devices, by reducing the number of MAC operations to near 300M [12].

5.4. Performance evaluation of RedZAP predictors

Fig. 10 illustrates the trade-off between MAC saving and accuracy considering the effect of the RedZAP predictors' accuracy on the amount of MAC savings. The observations are as follows:

Observation 9. In addition to the inference threshold, the training threshold also affected the amount of possible MAC reductions.

This observation introduces the training threshold as another knob, along with the inference threshold and mask type, for creating trade-off between MAC saving and accuracy. Among these knobs, the training threshold has the lowest level of flexibility in that when a RedZAP predictor is trained with a specific threshold, it cannot be altered during the inference phase. However, as shown, the trade-off can be provided by changing the inference threshold at the inference phase.

Observation 10. Given 1% top-1 accuracy loss as the target accuracy constraint, near 15% reduction in MAC operations was achieved using mask type A and training and inference simplification thresholds of 0.01 and 0.4, respectively.

This number does not seem to be comparable with the ZAP, which achieved near 30% MAC reduction under the same accuracy loss constraint, as reported in [68]. This is because the maximum amount of MAC savings possible under a specific accuracy loss is highly affected by the accuracy of predictors when identifying reducible convolutions. Although RedZAP and ZAP predictors were trained under the same configurations, i.e., 5 epochs and 50 K training data instances, the accuracy of RedZAP predictors was considerably below the accuracy of ZAP predictors, as shown in Fig. 11. This is because in contrast to the binary predictor ZAP, a multi-class predictor like RedZAP usually requires a more extensive training process. Therefore, to enable a fair comparison, the training of RedZAPs is in progress and the reports are available online[1] and are updated contentiously.

6. Conclusion

Factorizing MAC-intensive CONV operations into computationally low-cost pooling operations followed by 1×1 convolutions was found to enable 8% MAC savings in addition to MAC reduction opportunities offered by conventional zero and near-zero activation simplifications. The pooling functions were selected to be supported by existing CNN architectures, and exploring other alternatives can be considered as future work. Given a constraint on accuracy loss, the factorization is decided for each activation individually, using the proposed multi-class lightweight RedZAP predictor. The storage overhead of the proposed CONV factorization approach was calculated as negligible, although limiting the number of pooling functions that can be exploited.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was in part supported by a grant from IPM (No. 3-1401-28).

[1] https://drts.ut.ac.ir/RedZAP

References

[1] A. Pullini, F. Conti, D. Rossi, I. Loi, M. Gautschi, L. Benini, A heterogeneous multi-core system-on-chip for energy efficient brain inspired vision, in: 2016 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2016, p. 2910, http://dx.doi.org/10.1109/iscas.2016.7539213.
[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252, http://dx.doi.org/10.1007/s11263-015-0816-y.
[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778, http://dx.doi.org/10.1109/cvpr.2016.90.
[4] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, et al., MobileNets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv e-prints, arXiv:1704.04861.
[5] B. Moons, M. Verhelst, An energy-efficient precision-scalable convnet processor in 40-nm CMOS, IEEE J. Solid-State Circuits 52 (4) (2016) 903–914, http://dx.doi.org/10.1109/jssc.2016.2636225.
[6] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, NIPS, 2012, pp. 1097–1105.
[7] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv e-prints, arXiv:1409.1556.
[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2015, pp. 1–9, http://dx.doi.org/10.1109/cvpr.2015.7298594.
[9] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, ECCV, Springer, 2016, pp. 630–645, http://dx.doi.org/10.1007/978-3-319-46493-0_38.
[10] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2018, pp. 7132–7141, http://dx.doi.org/10.1109/cvpr.2018.00745.
[11] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016, arXiv e-prints, arXiv:1602.07360.
[12] X. Zhang, X. Zhou, M. Lin, J. Sun, ShuffleNet: An extremely efficient convolutional neural network for mobile devices, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2018, pp. 6848–6856, http://dx.doi.org/10.1109/cvpr.2018.00716.
[13] G. Huang, S. Liu, L. Van der Maaten, K.Q. Weinberger, CondenseNet: An efficient DenseNet using learned group convolutions, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2018, pp. 2752–2761, http://dx.doi.org/10.1109/cvpr.2018.00291.
[14] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, et al., Shift: A zero FLOP, zero parameter alternative to spatial convolutions, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2018, pp. 9127–9135, http://dx.doi.org/10.1109/cvpr.2018.00951.
[15] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2016, pp. 779–788, http://dx.doi.org/10.1109/cvpr.2016.91.
[16] J. Redmon, A. Farhadi, YOLO9000: Better, faster, stronger, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2017, pp. 7263–7271, http://dx.doi.org/10.1109/cvpr.2017.690.
[17] J. Redmon, A. Farhadi, YOLOv3: An incremental improvement, 2018, arXiv e-prints, arXiv:1804.02767.
[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, et al., SSD: Single shot multibox detector, in: European Conference on Computer Vision, ECCV, Springer, 2016, pp. 21–37, http://dx.doi.org/10.1007/978-3-319-46448-0_2.
[19] M. Wistuba, A. Rawat, T. Pedapati, A survey on neural architecture search, 2019, arXiv e-prints, arXiv:1905.01392.
[20] T. Elsken, J. Hendrik Metzen, F. Hutter, Neural architecture search: A survey, 2018, arXiv e-prints, arXiv:1808.05377.
[21] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, 2020, arXiv e-prints, arXiv:2006.02903.
[22] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, M. Sjödin, DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems, Microprocess. Microsyst. 73 (2020) 102989, http://dx.doi.org/10.1016/j.micpro.2020.102989.
[23] M. Loni, A. Zoljodi, D. Maier, A. Majd, M. Daneshtalab, M. Sjodin, B. Juurlink, R. Akbari, DenseDisp: Resource-aware disparity map estimation by compressing siamese neural architecture, in: 2020 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2020, http://dx.doi.org/10.1109/cec48606.2020.9185611.
[24] M. Loni, A. Zoljodi, S. Sinaei, M. Daneshtalab, M. Sjödin, NeuroPower: Designing energy efficient convolutional neural network architecture for embedded systems, in: Artificial Neural Networks and Machine Learning – ICANN 2019: Theoretical Neural Computation, Springer, 2019, pp. 208–222, http://dx.doi.org/10.1007/978-3-030-30487-4_17.

[25] Y. Zhou, S. Ebrahimi, S.Ö. Arık, H. Yu, H. Liu, G. Diamos, Resource-efficient neural architect, 2018, arXiv e-prints, arXiv:1806.07912.
[26] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, 2015, arXiv e-prints, arXiv:1510.00149.
[27] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, NIPS, 2015, pp. 1135–1143.
[28] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: Advances in Neural Information Processing Systems, NIPS, 2016, pp. 2074–2082.
[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient convnets, 2016, arXiv e-prints, arXiv:1608.08710.
[30] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: 2017 IEEE International Conference on Computer Vision, ICCV, IEEE, 2017, pp. 2736–2744, http://dx.doi.org/10.1109/iccv.2017.298.
[31] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: 2017 IEEE International Conference on Computer Vision, ICCV, IEEE, 2017, pp. 1389–1397, http://dx.doi.org/10.1109/iccv.2017.155.
[32] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, et al., Morphnet: Fast & simple resource-constrained structure learning of deep networks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2018, pp. 1586–1595, http://dx.doi.org/10.1109/cvpr.2018.00171.
[33] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, S. Han, AMC: AutoMl for model compression and acceleration on mobile devices, in: Proceedings of the European Conference on Computer Vision, ECCV, Springer, 2018, pp. 784–800, http://dx.doi.org/10.1007/978-3-030-01234-2_48.
[34] N. Nazari, M. Loni, M.E. Salehi, M. Daneshtalab, M. Sjodin, TOT-net: An endeavor toward optimizing ternary neural networks, in: 2019 22nd Euromicro Conference on Digital System Design, DSD, IEEE, 2019, http://dx.doi.org/10.1109/dsd.2019.00052.
[35] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, 2014, arXiv e-prints, arXiv:1412.6115.
[36] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: International Conference on Machine Learning, 2015, pp. 1737–1746.
[37] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental network quantization: Towards lossless cnns with low-precision weights, 2017, arXiv e-prints, arXiv:1702.03044.
[38] J. Wu, C. Leng, Y. Wang, Q. Hu, J. Cheng, Quantized convolutional neural networks for mobile devices, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2016, pp. 4820–4828, http://dx.doi.org/10.1109/cvpr.2016.521.
[39] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2016, arXiv e-prints, arXiv:1606.06160.
[40] D. Soudry, I. Hubara, R. Meir, Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights, in: Advances in Neural Information Processing Systems, NIPS, 2014, pp. 963–971.
[41] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: International Conference on Machine Learning, 2015, pp. 2285–2294.
[42] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in Neural Information Processing Systems, NIPS, 2015, pp. 3123–3131.
[43] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to 1 or -1, 2016, arXiv e-prints, arXiv:1602.02830.
[44] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-Net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision, ECCV, Springer, 2016, pp. 525–542, http://dx.doi.org/10.1007/978-3-319-46493-0_32.
[45] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, arXiv e-prints, arXiv:1503.02531.
[46] A. Romero, N. Ballas, S. Ebrahimi Kahou, A. Chassang, C. Gatta, Y. Bengio, FitNets: Hints for thin deep nets, 2014, arXiv e-prints, arXiv:1412.6550.
[47] A.K. Balan, V. Rathod, K.P. Murphy, M. Welling, Bayesian dark knowledge, in: Advances in Neural Information Processing Systems, NIPS, 2015, pp. 3438–3446.
[48] P. Luo, Z. Zhu, Z. Liu, X. Wang, X. Tang, Face model compression by distilling knowledge from neurons, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Vol. 30, 2016, pp. 3560–3566.
[49] T. Chen, I. Goodfellow, J. Shlens, Net2net: Accelerating learning via knowledge transfer, 2015, arXiv e-prints, arXiv:1511.05641.
[50] S. Zagoruyko, N. Komodakis, Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, 2016, arXiv e-prints, arXiv:1612.03928.
[51] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, et al., DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, ACM SIGARCH Comput. Architect. News 42 (1) (2014) 269–284, http://dx.doi.org/10.1145/2654822.2541967.
[52] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, et al., ShiDianNao: Shifting vision processing closer to the sensor, in: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ACM, 2015, pp. 92–104, http://dx.doi.org/10.1145/2749469.2750389.
[53] A. Yasoubi, R. Hojabr, M. Modarressi, Power-efficient accelerator design for neural networks using computation reuse, IEEE Comput. Architect. Lett. 16 (1) (2016) 72–75, http://dx.doi.org/10.1109/lca.2016.2521654.
[54] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, C. Fletcher, UCNN: Exploiting computational reuse in deep neural networks via weight repetition, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 674–687, http://dx.doi.org/10.1109/isca.2018.00062.
[55] M. Riera, J.-M. Arnau, A. González, Computation reuse in DNNs by exploiting input similarity, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 57–68, http://dx.doi.org/10.1109/isca.2018.00016.
[56] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, A. Moshovos, Bit-pragmatic deep neural network computing, in: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2017, pp. 382–394, http://dx.doi.org/10.1145/3123939.3123982.
[57] H. Mahdiani, A. Khadem, A. Ghanbari, M. Modarressi, F. Fattahi-Bayat, M. Daneshtalab, δnn: Power-efficient neural network acceleration using differential weights, IEEE Micro 40 (1) (2019) 67–74, http://dx.doi.org/10.1109/mm.2019.2948345.
[58] Y. Lin, C. Sakr, Y. Kim, N. Shanbhag, PredictiveNet: An energy-efficient convolutional neural network via zero prediction, in: 2017 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2017, pp. 1–4, http://dx.doi.org/10.1109/iscas.2017.8050797.
[59] M. Song, J. Zhao, Y. Hu, J. Zhang, T. Li, Prediction based execution on deep neural networks, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 752–763, http://dx.doi.org/10.1109/isca.2018.00068.
[60] J. Chang, Y. Choi, T. Lee, J. Cho, Reducing MAC operation in convolutional neural network with sign prediction, in: 2018 International Conference on Information and Communication Technology Convergence, ICT, IEEE, 2018, pp. 177–182, http://dx.doi.org/10.1109/ictc.2018.8539530.
[61] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R.K. Gupta, H. Esmaeilzadeh, SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 662–673, http://dx.doi.org/10.1109/isca.2018.00061.
[62] J. Zhu, J. Jiang, X. Chen, C.-Y. Tsui, SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity, in: 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2018, pp. 241–244, http://dx.doi.org/10.23919/date.2018.8342010.
[63] G. Shomron, U. Weiser, Spatial correlation and value prediction in convolutional neural networks, IEEE Comput. Architect. Lett. 18 (1) (2018) 10–13, http://dx.doi.org/10.1109/lca.2018.2890236.
[64] X. Dong, J. Huang, Y. Yang, S. Yan, More is less: A more complicated network with less inference complexity, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2017, http://dx.doi.org/10.1109/cvpr.2017.205.
[65] M. Figurnov, A. Ibraimova, D.P. Vetrov, P. Kohli, PerforatedCNNs: Acceleration through elimination of redundant convolutions, in: Advances in Neural Information Processing Systems, NIPS, 2016, pp. 947–955.
[66] Y. Huan, Y. Qin, Y. You, L. Zheng, Z. Zou, A multiplication reduction technique with near-zero approximation for embedded learning in IoT devices, in: 2016 29th IEEE International System-on-Chip Conference, SOCC, IEEE, 2016, pp. 102–107, http://dx.doi.org/10.1109/socc.2016.7905445.
[67] C. Kim, D. Shin, B. Kim, J. Park, Mosaic-CNN: A combined two-step zero prediction approach to trade off accuracy and computation energy in convolutional neural networks, IEEE J. Emerg. Sel. Top. Circuits Syst. 8 (4) (2018) 770–781, http://dx.doi.org/10.1109/jetcas.2018.2865006.
[68] G. Shomron, R. Banner, M. Shkolnik, U. Weiser, Thanks for nothing: Predicting zero-valued activations with lightweight convolutional neural networks, in: European Conference on Computer Vision, ECCV, Springer, 2020, pp. 234–250, http://dx.doi.org/10.1007/978-3-030-58607-2_14.
[69] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, K. Roy, IMPACT: Imprecise adders for low-power approximate computing, in: IEEE/ACM International Symposium on Low Power Electronics and Design, IEEE, 2011, http://dx.doi.org/10.1109/islped.2011.5993675.
[70] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, A. Yakovlev, Significance-driven logic compression for energy-efficient multiplier design, IEEE J. Emerg. Sel. Top. Circuits Syst. 8 (3) (2018) 417–430, http://dx.doi.org/10.1109/jetcas.2018.2846410.
[71] S.S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, K. Roy, Energy-efficient neural computing with approximate multipliers, ACM J. Emerg. Technol. Comput. Syst. 14 (2) (2018) 1–23, http://dx.doi.org/10.1145/3097264.
[72] C. Guo, L. Zhang, X. Zhou, W. Qian, C. Zhuo, A reconfigurable approximate multiplier for quantized CNN applications, in: 2020 25th Asia and South Pacific Design Automation Conference, ASP-DAC, IEEE, 2020, http://dx.doi.org/10.1109/asp-dac47756.2020.9045176.

[73] V. Sze, Y.-H. Chen, T.-J. Yang, J.S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE 105 (12) (2017) 2295–2329, http://dx.doi.org/10.1109/jproc.2017.2761740.
[74] A. Fog, Instruction Tables: Lists of Instruction Latencies, Throughputs and Micro-Operation Breakdowns for Intel, AMD and Via CPUs, Technical University of Denmark, 2019.
[75] A. Yoosefi, M. Kargahi, Improving energy-efficiency of CNNs via prediction of reducible convolutions for energy-constrained IoT devices, in: 2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies, RTEST, IEEE, 2020, pp. 1–9, http://dx.doi.org/10.1109/rtest49666.2020.9140143.
[76] Y.-L. Boureau, J. Ponce, Y. LeCun, A theoretical analysis of feature pooling in visual recognition, in: Proceedings of the 27th International Conference on Machine Learning, ICML-10, 2010, pp. 111–118.
[77] A. Krizhevsky, V. Nair, G. Hinton, CIFAR-10 and CIFAR-100 datasets, 2009, p. 1, URL: https://www.cs.toronto.edu/kriz/cifar.html, 6.
[78] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv e-prints, arXiv:1412.6980.

A. Yoosefi received the B.Sc. degree from the Shahid Bahonar University of Kerman, Kerman, Iran, in 2013, and the M.Sc. degree from the Graduate University of Advanced Technology, Kerman, Iran, in 2016. He is currently a Ph.D. candidate at the School of ECE, College of Eng., University of Tehran, Tehran, Iran. He is also a research member of the Dependable/Distributed Real-Time Systems (DRTS) Research Laboratory, at the University of Tehran. His current research interests include Resource-Constrained Deep-Learning, RealTime Embedded Systems, Distributed Computing, and Reconfigurable Computing. You can contact him at a.yoosefi@ut.ac.ir.