Thanks for Nothing: Predicting Zero-Valued Activations with Lightweight Convolutional Neural Networks

Gil Shomron[1], Ron Banner[2], Moran Shkolnik[1,2], and Uri Weiser[1]

[1] Faculty of Electrical Engineering — Technion, Haifa, Israel
{gilsho@campus, uri.weiser@ee}.technion.ac.il
[2] Habana Labs — An Intel Company, Caesarea, Israel
{rbanner, mshkolnik}@habana.ai

Abstract. Convolutional neural networks (CNNs) introduce state-of-the-art results for various tasks with the price of high computational demands. Inspired by the observation that spatial correlation exists in CNN output feature maps (ofms), we propose a method to dynamically predict whether ofm activations are zero-valued or not according to their neighboring activation values, thereby avoiding zero-valued activations and reducing the number of convolution operations. We implement the zero activation predictor (ZAP) with a lightweight CNN, which imposes negligible overheads and is easy to deploy on existing models. ZAPs are trained by mimicking hidden layer ouputs; thereby, enabling a parallel and label-free training. Furthermore, without retraining, each ZAP can be tuned to a different operating point trading accuracy for MAC reduction.

Keywords: Convolutional neural networks, dynamic pruning

1 Introduction

In the past decade, convolutional neural networks (CNNs) have been adopted for numerous applications [39][37][27], introducing state-of-the-art results. Despite being widely used, CNNs involve a considerable amount of computations. For example, classification of a 224x224 colored image requires billions of multiply-accumulate (MAC) operations [4][42]. Such computational loads have many implications, from execution time to power and energy consumption of the underlying hardware.

CNN output feature maps (ofms) have been observed to exhibit spatial correlation, i.e., adjacent ofm activations share close values [32][38]. This observation is particularly true for zero-valued activations, as it is common practice to use the ReLU activation function [33], which squeezes all negative values to zero. If it were possible to predict which of the convolution operations will result in a negative value, they could be skipped, their corresponding ofm activations could be set to zero, and many multiply-accumulate (MAC) operations could be saved.

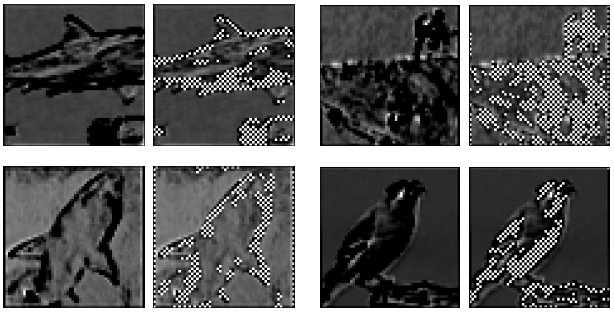

Fig. 1. Exploiting spatial correlation for zero-value prediction of ofm activations with CNN-based predictor. Bright white pixels represent a predicted zero-valued ofm activation.

Prediction mechanisms are at the heart of many general-purpose processors (GPPs), leveraging unique application characteristics, such as code semantics and temporal locality, to predict branch prediction outcomes and future memory accesses, for example. Prediction mechanisms may similarly be employed for CNNs. In this paper, we propose a prediction method for CNNs that dynamically classifies ofm activations as zero-valued or non-zero-valued by leveraging the spatial correlation of ofms. The zero activation predictor (ZAP) works in three-steps: first, only a portion of the ofm is fully computed; then, the remaining activations are classified as zero-valued or non-zero-valued using a lightweight convolutional neural network; and finally, the predicted non-zero-valued activations are computed while the zero-valued activations are skipped, thereby saving entire convolution operations (Figure 1). ZAP imposes negligible overheads in terms of computations and memory footprint, it may be plugged into pretrained models, it is trained quickly, in parallel, and does not require labeled data.

ZAP may be considered as a CNN-based, dynamic, unstructured, and magnitude-based ofm activations pruning strategy. However, as opposed to many pruning techniques, ZAP is tunable on-the-fly. Therefore, ZAP captures a wide range of operating points, trading accuracy for MAC savings. For example, by strictly focusing on zero-valued ofm activations, ZAP can capture a range of operating points that does not require retraining. ZAP is also capable of pruning when set to high speculation levels, as more mispredictions of non-zero-valued activations as zero-valued activations take place. Interestingly, we observe that these mispredictions usually occur with small activation values, which is practically a magnitude-based pruning strategy.

This paper makes the following contributions:

- **Zero activation predictor (ZAP).** We introduce a dynamic, easy to deploy, CNN-based zero-value prediction method that exploits the spatial correlation of output feature map activations. Compared with conventional convolution layers, our method imposes negligible overheads in terms of both computations and parameters.

- **Trade-off control.** We estimate the model's entire accuracy-to-savings trade-off curve with mostly local statistics gathered by each predictor. This provides a projection of the model and the predictor performance for any operating point.
- **Accuracy to error linearity.** We show, both analytically and empirically, that the entire model accuracy is linear with the sum of local misprediction errors, assuming they are sufficiently small.
- **Local tuning given global constraints.** We consider layers variability by optimizing each ZAP to minimize the model MAC operations subject to a global error budget.

2 ZAP: Zero Activation Predictor

In this section, we describe our prediction method and its savings potential. We analyze its overheads in terms of computational cost and memory footprint and show that both are negligible compared with conventional convolution layers.

2.1 Preliminary

A convolution layer consists of a triplet $\langle X_i, X_o, W \rangle$, where X_i and X_o correspond to the input and output activation tensors, respectively, and W corresponds to the set of convolution kernels. Each activation tensor is a three-dimensional tensor of size $w \times h \times c$, where w, h and c represent the tensor width, height, and depth, respectively. For convenience, the dimensions of X_i and X_o are denoted by $[w_i \times h_i \times c_i]$ and $[w_o \times h_o \times c_o]$. Finally, the set of convolution kernels W is denoted by a four-dimensional tensor $[k \times k \times c_i \times c_o]$, where k is the filter width and height. The filter width and height are not necessarily equal, but it is common practice in most conventional CNNs to take them a such. Given the above notations, each output activation value $X_o[x, y, z]$ is computed from the input tensor X_i and weights W as follows:

$$X_o(x, y, z) = \sum_{i,j=0}^{k-1} \sum_{c=0}^{c_i-1} X_i[x + i, y + j, c] \cdot W[i, j, c, z], \tag{1}$$

where the bias term is omitted for simplicity's sake. We use the above notations throughout this paper.

2.2 Three-Step Method

Conventional convolution layers compute their ofm by applying Equation (1) for each $[x, y, z]$. It has been observed that ofms accommodate many zero-valued activations due to the widespread usage of the ReLU activation function [34][42][2]. For example, with ResNet-18 [13], almost 50% of the ofm activations are zero-valued; therefore, 50% of the MAC operations may be potentially skipped. Moreover, ofm activations exhibit spatial correlation [32][38], meaning that a group of activation values testifies about other adjacent activation values.

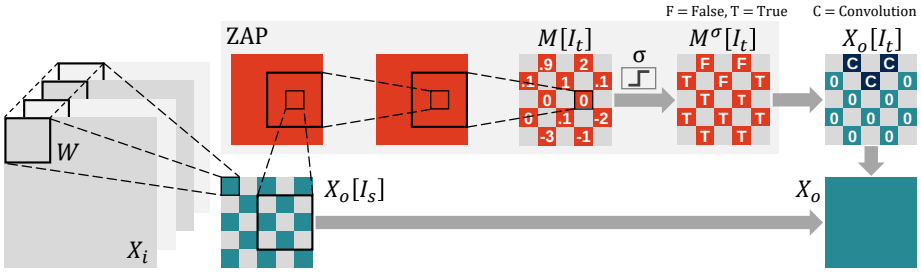

Fig. 2. Illustration of a single ofm channel convolution with ZAP. $X_o[I_s]$ is computed based on a pre-defined pattern. $X_o[I_s]$ is then subjected to ZAP, which produces a prediction map $M[I_t]$, followed by a thresholding step to form the binary prediction map $M^\sigma[I_t]$. $X_o[I_t]$ is created according to $M^\sigma[I_t]$ — a portion of the I_t ofm activations are predicted as zero-valued and skipped, whereas the rest are computed using Equation (1).

We suggest a three-step dynamic prediction mechanism that locates zero-valued ofm activations by exploiting the spatial correlation inherent in CNNs to potentially skip them and reduce the computational burden of the model. Given an ofm, X_o, we divide its indices into two subsets (I_s, I_t) according to a pre-defined pattern. Then, the following three steps are carried out (as illustrated in Figure 2): (i) the values that belong to indices I_s are fully computed in the conventional manner, resulting in a sparse ofm, $X_o[I_s]$; (ii) this partial ofm $(X_o[I_s])$ is passed through our predictor to yield a binary map, $M^\sigma[I_t]$, which is used to predict the zero values in $X_o[I_t]$; and (iii) all values predicted to be non-zero by $M^\sigma[I_t]$ are computed. We describe this process in detail next.

Computation pattern. $X_o[I_s]$ is computed based on a pre-defined pattern as depicted in Figure 3. Since our predictor exploits spatial correlation, we partially compute the ofm so that activations that are not computed in the first step will reside next to a number of computed activations. We use α to denote the ratio between the number of activations computed in the partial ofm and the ofm dimensions. This may be formally formulated as $\alpha \equiv \frac{|I_s|}{w_o h_o c_o}$, where $|I_s|$ is the set cardinality.

By decreasing α, less ofm activations are computed prior to the prediction, which may potentially lead to the saving of more operations. For example, for $\alpha = 40\%$, 60% of the activations may potentially be saved. Less prior data about the ofm may, however, lead to higher misprediction rates, which in turn may decrease model accuracy.

Prediction process via lightweight CNN. Given the partial ofm, $X_o[I_s]$, our goal is to produce an output computation mask that predicts which of the remaining activations are zero-valued and may be skipped. Recall that ofm activations are originally computed using Equation (1) and so the prediction process must involve less MAC operations than the original convolution operation with as minimal memory footprint as possible.

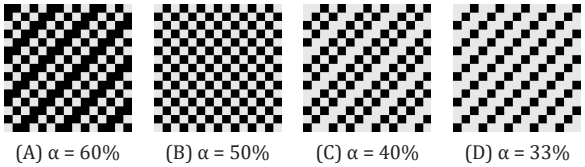

Fig. 3. Partial ofm convolution patterns. Black pixels represent computed activations.

We use a CNN to implement ZAP. We exploit the spatial correlation inherent in CNNs [32][38][21][9] and use only depthwise convolution (DW-CONV) layers [16], i.e., only spatial filters with no depth ($k \times k \times 1 \times c_o$). As such, we obtain a lightweight model in terms of both MAC operations and parameters (further analyzed in Section 2.3). Our CNN comprises a 3x3 DW-CONV layer followed by a batch normalization (BN) layer [20] followed by ReLU, twice. DW-CONV layer padding and stride are both defined as 1 to achieve equal dimensions throughout the CNN predictor. During training, the last ReLU is capped at 1 [24], whereas during inference we discard the last ReLU activation and use a threshold.

Thresholding. Altough ZAP is trained to output a binary classifier (described in Section 4.1), ZAP naturally outputs $M[I_t]$, which is not a strict binary mask but rather a range of values that corresponds to a prediction confidence [11][43] of whether the ofm activation is zero-valued or not. To binarize ZAP's output, we define a threshold σ that sets the level of prediction confidence; therefore, $M^\sigma[I_t] = M[I_t] > \sigma$ (boolean operation).

According to $M^\sigma[I_t]$, part of the ofm activations in $X_o[I_t]$ are predicted to be non-zero-valued and are computed, whereas the others are predicted to be zero-valued and are skipped. When an ofm activation is predicted to be zero-valued, two types of misprediction may occur. First, an actual zero-valued activation may be predicted as a non-zero-valued activation and so redundant MAC operations may take place. Second, an actual non-zero-valued activation may be predicted as zero. The latter misprediction increases the model error, potentially decreasing model accuracy.

The motivation behind ZAP is clear — reducing computations by skipping convolution operations. Its impact on the model accuracy, number of computations, and amount of parameters has yet, however, to be discussed. We next discuss the overhead of ZAP, and in Section 4, we show empirically how ZAP affects the accuracy of different model architectures.

2.3 Overhead Analysis

ZAP is a CNN by itself, which means that it introduces additional computations and parameters. To be beneficial, it must execute less operations per ofm activation than the original convolution operation. A conventional convolution layer requires $k^2 c_i$ MAC operations per ofm activation. On the other hand, the number of MAC operations required by a two-layered DW-CONV ZAP for a single ofm activation is K^2, where K is the filter width and height. Note that ZAP's

first layer needs only to consider $|I_s|$ values for computation since, according to the pre-defined pattern, the remaining $|I_t|$ values are zero; the second layer needs only to compute $|I_t|$ ofm activations.

Compared with a standard convolution operation, and for the case of $K = k$,

$$\frac{\text{ZAP ops.}}{\text{standard convolution ops.}} = \frac{1}{c_i}\,. \tag{2}$$

c_i is usually greater than 10^2 [13][40][26], in which case $1/c_i \approx 0$ and ZAP overhead is, therefore, negligible.

Regarding the parameters, a conventional convolution layer requires $k^2 c_i c_o$ parameters and a two-layered DW-CONV requires $2K^2 c_o$ parameters. Given c_i conventional sizes and $K = k$, ZAP's memory footprint is negligible as well.

3 Trade-Off Control

The threshold hyperparameter, σ, represents the prediction confidence of whether an ofm activation is zero or non-zero. Users should, however, address the accuracy and MAC reduction terms rather than using σ, since it is not clear how a specific σ value affects accuracy and savings. In this section, we show that, given some statistics, we can estimate the entire model accuracy and predictor MAC savings, thereby avoiding the need to address the threshold value directly and providing the user with an accuracy-MAC savings trade-off control "knob".

3.1 Accuracy, MAC Savings, and Threshold

Accuracy and threshold. Consider a DNN with L layers. Each layer i comprises weights w_i, an ifm x_i, and an ofm y_i. When predictions are avoided, layer $i+1$ input, x_{i+1}, is given by

$$x_{i+1} = y_i = \max(x_i w_i, 0)\,, \tag{3}$$

where $\max(\cdot, 0)$ is the ReLU activation function.

Our predictor is not perfect and may zero out non-zero elements with a small probability ε_i. Therefore, non-zero inputs to layer $i+1$ have a probability ε_i of becoming zero and a probability $1-\varepsilon$ of remaining y_i. Using y_i^{π} to denote the prediction of output y_i, we obtain the following error in the expected activation:

$$E(y_i^{\pi}) = (1-\varepsilon_i) \cdot E(y_i)\,. \tag{4}$$

In other words, the prediction at each layer i introduces a multiplicative error of $(1-\varepsilon_i)$ with respect to the true activation. This multiplicative error builds up across layers. For an L-layer network, the expected network outputs are scaled down with respect to the true network outputs as follows:

$$\text{Scale Error} = \prod_{i=1}^{L}(1-\varepsilon_i)\,. \tag{5}$$

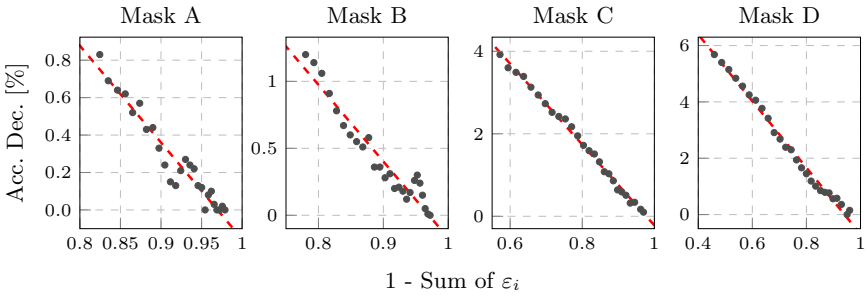

Fig. 4. AlexNet + CIFAR-100 top-1 accuracy decrease as a function of $1-\sum_{i=1}^{L}\varepsilon_i(\sigma)$. Each dot represents a measurement with a different threshold: the leftmost and rightmost thresholds are 0 and 0.5, respectively. Measurements were taken in steps of 0.02.

Note that for a sufficiently small ε, $1-\varepsilon = e^{-\varepsilon}$. Therefore, assuming sufficiently small misprediction probabilities, $\{\varepsilon_i\}$, we obtain the following expression

$$\begin{aligned} \text{Scale Error} &= \prod_{i=1}^{L}(1-\varepsilon_i) \approx \prod_{i=1}^{L} e^{-\varepsilon_i} \\ &= e^{-\sum_{i=1}^{L}\varepsilon_i} \approx 1-\sum_{i=1}^{L}\varepsilon_i\,, \end{aligned} \tag{6}$$

where the approximations can easily be extracted from a Taylor expansion to e^{-x}. Equation (6) shows that the final error due to small mispredictions accumulates along the network in a *linear* fashion when the errors due to mispredictions are small enough.

Denoting the output of layer i for a threshold σ by $y_i^{\pi,\sigma}$, the error is associated with a threshold σ as follows:

$$\begin{aligned} \text{Scale Error}\,(\sigma) &\approx 1-\sum_{i=1}^{L}\varepsilon_i(\sigma) \\ &= 1-\sum_{i=1}^{L}\left(1-\frac{\sum_{x,y,z} y_i^{\pi,\sigma}[x,y,z]}{\sum_{x,y,z} y_i[x,y,z]}\right)\,, \end{aligned} \tag{7}$$

where Equation (4) is used for the last transition. Figure 4 shows that this analytical observation is in good agreement with our empirical results.

Ideally, we would like to have a scale error that is as close as possible to 1 so as to avoid shifting the network statistics from the learned distribution. When the scale error is 1, network outputs remain unchanged and no accuracy degradation associated with mispredictions occurs. Yet, as σ increases, more ofm activations are predicted as zero-valued, decreasing the scale error below 1, as exhibited by Equation (7). This scale error decreases monotonically with σ and introduces an inverse mapping, which enables us to define a threshold value for any desired accuracy degradation.

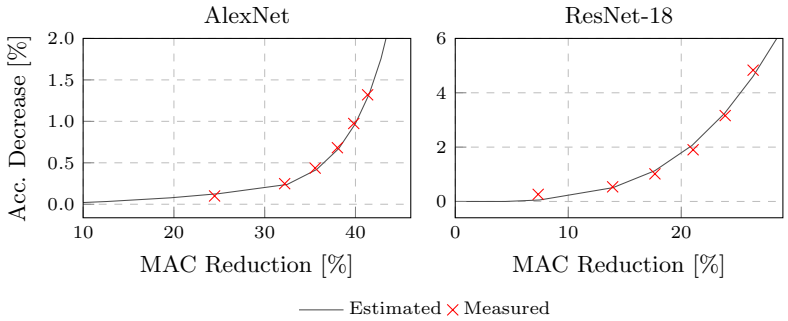

Fig. 5. Estimated top-5 accuracy-MAC savings trade-off curve using mask B and ILSVRC-2012 dataset. The measured operating points correspond to thresholds of 0 to 0.5, in steps of 0.1.

MAC savings and threshold. Recall that $M[I_t]$ represents the collection of values predicted by ZAP for a given layer before applying the threshold. For readability, assume $m(x)$ is the probability density function of $M[I_t]$ values (i.e., continuous). Then, the number of ofm activations predicted as zeros relative to I_t can be expressed as follows:

$$\text{Zero Prediction Rate}\,(\sigma) = \int_{-\infty}^{\sigma} m(x)dx\,. \tag{8}$$

To achieve the actual MAC reduction in layer i, the layer dimensions, α, and ZAP overhead should be considered. Clearly, the total MAC reduction equals the sum of the contributions from all layers. Note that the zero prediction rate, and therefore the MAC reduction, increases monotonically, and for a certain range it may be considered a strictly monotonic function. Inverse mapping from MAC reduction to σ is, therefore, possible.

Putting it all together. Given the derivations so far, it is possible to make an *a priori* choice of any operating point, given estimates about the desired accuracy and MAC reduction, without directly dealing with the threshold. Assume a pre-processing step that consists of collecting statistical information about the prediction values $M[I_t]$ and at least two accuracy measurements of the entire model, to obtain the linear accuracy-error relationship. With this statistical information, we can effectively estimate the desired operating point, as demonstrated in Figure 5.

3.2 Non-Uniform Threshold

Thus far, for the sake of simplicity, σ was set uniformly across all layers. Layers, however, behave differently, and so ZAP error and savings may differ between layers for a given threshold. This is evident in Figure 6 in which we present the error and total MAC operations of four layers in ResNet-18. Notice how the

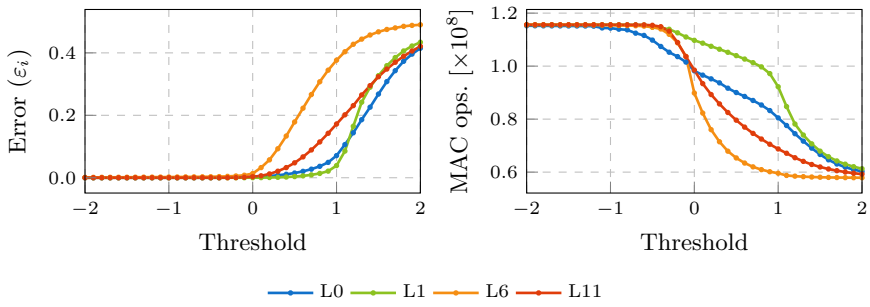

Fig. 6. ResNet-18 + ILSVRC-2012 example of different layers behavior in terms of error and MAC operations as a function of ZAP threshold.

error of layer 6 (L6) increases earlier than the other layers, for example. Ideally, given L layers, we would like to choose a threshold per layer, σ_i, to save as many MAC operations as possible, given an error (or accuracy) constraint, ϵ, i.e.,

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{L} \text{MAC ops.}_i\,(\sigma_i) \\ \text{subject to} \quad & \sum_{i=1}^{L} \varepsilon_i(\sigma_i) \leq \epsilon \propto \text{Accuracy}. \end{aligned} \tag{9}$$

We use curve fitting to define a parameterized sigmoid function for the error and for the MAC operations of each layer and apply standard non-linear optimization methods to solve Equation (9) (see supplementary material for curve fitting results). It is worth mentioning that Equation (9) can also be written the other way around, that is, minimizing the error given a computation budget.

4 Experiments

In this section, we evaluate ZAP performance in terms of MAC savings and accuracy degradation using various model architectures and datasets, and compare our results to previous work.

4.1 ZAP Training

ZAPs are deployed at each desired convolution layer and are trained independently. By training in isolation [14][8], ZAPs can be plugged into a model without altering its architecture and trained parameters and may be trained in parallel. First, a batch is fed forward bypassing all predictors. During the feedforward phase, each ZAP saves a copy of its local input feature map (ifm) and corresponding local ofm. Then, each ZAP computes its $M[I_t]$, using its ifm followed

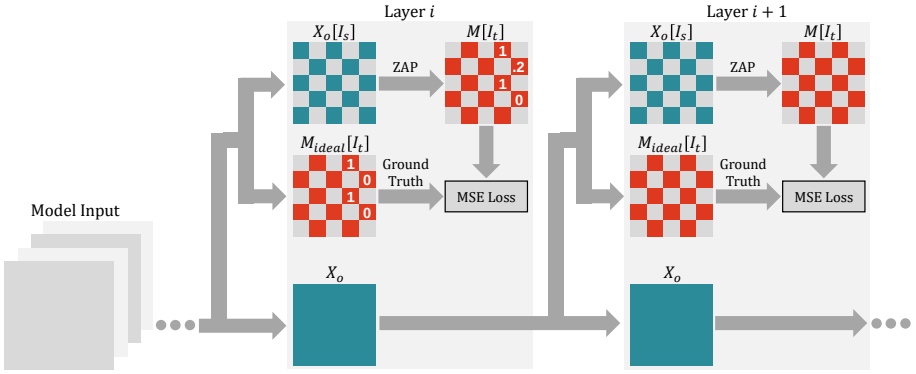

Fig. 7. Illustration of ZAP training. Each ZAP is trained independently in a teacher-student manner, enabling a parallel label-free training. The ground truth is the original layer ofm after a binary threshold operation (>0). The CNN predictor is used with the ReLU activation function capped at 1.

by a ReLU activation function which is capped at 1. The ground truth, M_{ideal}, of each ZAP is its local ofm passed through a zero-threshold boolean operation at indices I_t. Finally, the MSE loss is used to minimize each of the predictor errors, as follows:

$$\min \sum_{(x,y,z)\in I_t} \left(M - M_{ideal}\right)^2 . \tag{10}$$

Notice that no labeled data is needed. ZAP training is illustrated in Figure 7.

4.2 Experimental Setup

We evaluated our method using CIFAR-100 [25] and ILSVRC-2012 [36] datasets, and AlexNet [26], VGG-16 [40], and ResNet-18 [13] CNN architectures. The source code is publicly available[3].

ZAPs were trained using the Adam [22] optimizer for 5 epochs. When the ILSVRC-2012 dataset was used for ZAPs training, only 32K training set images were used. After ZAPs were trained, we recalibrated the running mean and variance of the model BN layers; this is not considered fine-tuning since it does not involve backpropagation. BN recalibration was noticeably important with ResNet-18, which has multiple BN layers. When fine-tuning was considered, it was limited to 5 epochs with CIFAR-100 and to 1 epoch with ILSVRC-2012. We did not deploy ZAPs on the first layers of any of the models, since it is not beneficial in terms of potential operation savings.

AlexNet with CIFAR-100 was trained from scratch using the original hyper-parameters, achieving top-1 accuracy of 64.4%. AlexNet, VGG-16, and ResNet-18 with ILSVRC-2012 were used with the PyTorch pretrained parameters, achiev-

[3] https://github.com/gilshm/zap

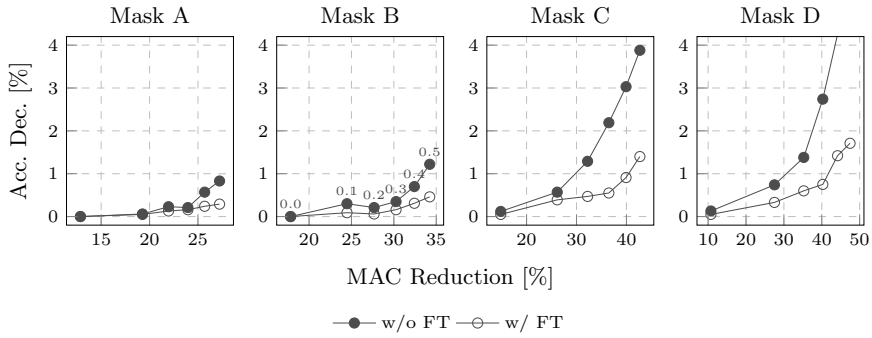

Fig. 8. Demonstrating the different operating points of AlexNet + CIFAR-100 with (w/) and without (w/o) fine-tuning (FT). Accuracy measurements are top-1. Threshold range is set between 0 to 0.5, in steps of 0.1. Each measurement corresponds to a threshold, as presented in "Mask B" plot.

ing top-1/top-5 accuracies of 56.5%/79.1%, 71.6%/90.6%, and 69.8%/89.1%, respectively.

We experimented with four different prediction patterns (Figure 3). The same prediction pattern was used across all layers and channels. Mixing patterns in layer and channel granularity is an option we leave for future work.

Throughout this section, MAC reduction is defined as $(1 - \text{after/before})$ and accuracy degradation is defined as $(\text{before} - \text{after})$. When discussing MAC reduction, we consider only the relative savings from convolution layers.

4.3 CIFAR-100

Operating points. We demonstrate different operating points as measured with different masks and uniform thresholds across all layers with AlexNet using the CIFAR-100 dataset. For each operating point, we report the entire model top-1 accuracy degradation and MAC operation savings, with and without fine-tuning (Figure 8). For example, considering a 1% top-1 accuracy degradation cap, ZAP achieves a 32.4% MAC reduction with 0.7% accuracy degradation at mask B with a 0.4 threshold. In addition, by fine-tuning the model, our predictor achieves 40.3% MAC reduction with 0.8% accuracy degradation at mask D with a 0.3 threshold.

Masks with greater α values lead to ofms with relatively more computed activations, i.e., larger $|I_s|$. We observe that these masks show better accuracy results in the low degradation region (e.g., mask A versus mask D at 20% MAC reduction), whereas for higher MAC reduction requirements, masks with lower α values are preferred. In order to achieve high MAC reductions with the former masks (for example, mask A), σ would have to be cranked up. Since the prediction potential of these masks is low to begin with (for example, the best-case scenario with mask A is 40% activations savings with mask A), high thresholds

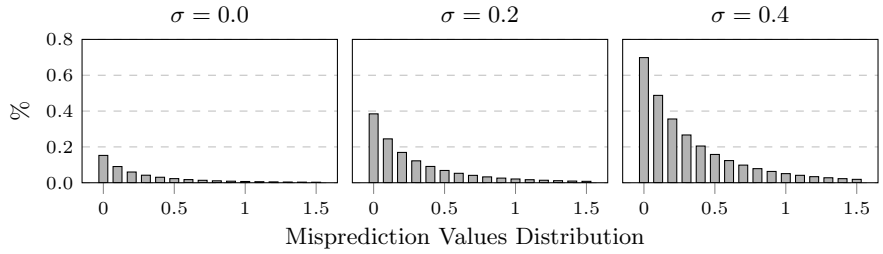

Fig. 9. AlexNet + CIFAR-100 histograms of non-zero values that were predicted as zero using mask B and different uniform thresholds.

will lead to pruning of relatively significant values and, as a result, to significant accuracy degradation. On the other hand, for conservative MAC reductions, these masks are preferred, since their speculation levels are lower and prediction confidence is higher.

Misprediction breakdown. The model accuracy is not solely dependent on *how many* non-zeros were predicted as zero, but also *which* values were zeroed out. The notion that small values within DNN are relatively ineffectual is at the core of many pruning techniques. In Figure 9 we present the mispredicted activation values distribution, normalized to the number of ofm activations for value steps of 0.1. For example, when using $\sigma = 0$, 0.15% of ofm activations with original values between 0 to 0.1 were zeroed out. Increasing the threshold level increases the number of predicted and mispredicted ofm activations. It is apparent that insignificant values are more prone to be mispredicted — it is easier to be mistaken about zeroing out a small value than a large value — attesting to the correlation between σ and the prediction confidence level.

4.4 ILSVRC-2012

Comparison with previous work. In Table 1 we compare our method with previous works. Shomron and Weiser [38], and Kim et al. [21] focus on predicting zero-valued activations according to nearby zero-valued activations. Figurnov et al. [9], on the other hand, mainly use "nearest neighbor" to interpolate missing activation values. Dong et al. [7] present a network structure with embedded low-cost convolution layers that act as zero-valued activation masks for the original convolution masks.

Using AlexNet and VGG-16, our method shows better results across the board (besides a small difference of 0.07 with VGG-16 top5 compared to Kim et al.) — for more MAC savings, a smaller accuracy drop is observed. Regrading ResNet-18, our method shows better results compared with Shomron and Weiser but falls short when compared with Dong et al. Notice though that the method Dong et al. introduced involves an entire network training with *hundreds* of epochs. Therefore, even though there is some resemblance between our method and theirs, the results themselves are not necessarily comparable.

Table 1. MAC reduction and accuracy with and without fine-tuning (ft) compared with previous work. Thresholds are set by solving Equation 9 (see supplementary material for execution details). The accuracy columns represent the decrease in accuracy. The MAC columns represent the MAC operations reduction. The minus ('-') symbol represents unpublished data.

Net	Related Work						Ours				
	Paper	Top1	ft	Top5	ft	MAC	Top1	ft	Top5	ft	MAC
AlexNet	Figurnov et al.	-	-	8.50	2.0	50.0%	4.63	1.97	3.04	1.17	51.1%
	Kim et al.	0.48	-	0.40	-	28.6%	0.34	0.33	0.24	0.14	32.4%
	Shomron et al.	4.00	1.6	2.90	1.3	37.8%	1.28	0.78	0.84	0.42	38.0%
VGG-16	Figurnov et al.	-	-	15.6	1.1	44.4%	11.02	1.27	7.02	0.71	44.5%
	Kim et al.	0.68	-	0.26	-	25.7%	0.54	0.24	0.33	0.11	26.2%
	Shomron et al.	3.60	0.7	2.00	0.4	30.7%	1.71	0.31	0.87	0.19	31.3%
ResNet-18	Dong et al. *	-	3.6	-	2.3	34.6%	12.35	7.22	8.37	4.66	34.0%
	Shomron et al.	11.0	2.7	7.60	1.7	22.7%	2.96	1.86	1.70	1.13	23.8%

All MAC saving measurements reported in Table 1 are theoretical, that is, they are not actual speedups. However, it is apparent that the *potential* of ZAP is greater than the other previous works. We discuss related hardware implementations and related work next.

5 Discussion and Related Work

Hardware. It is not trivial to attain true benefits from mainstream compute engines, such as GPUs, when dealing with compute intensive tasks, such as CNNs, and using a conditional compute paradigm of which only a portion of the MAC operations are conducted and the rest are performed or skipped according to the previously computed results. However, the implementation of CNN *accelerators*, which are capable of doing so and gain performance, has already been demonstrated. Kim et al. [21] present an architecture based on SCNN [34] which is capable of skipping computations of entire ofm activations based on already computed zero-valued ofm activations. Moreover, Hua et al. [18], Akhlaghi et al. [1] and Asadikouhanjani et al. [3] propose hardware that is capable of saving a portion of the MAC operations needed per ofm activation based on the accumulated partial sum. Akhlaghi et al. also suggest a method to do so in a speculative manner.

Speculative execution. GPPs make extensive use of speculative execution [15]. They leverage unique application characteristics, such as code semantics and temporal locality, to better utilize the GPP's internal structure to achieve better performance. Researchers have also studied the speculative execution of CNNs to decrease their compute demands. Song et al. [41], Lin et al. [30], and Chang et al. [5] predict whether an entire convolution result is negative according to a partial result yielded by the input MSB bits. Huan et al. [19] avoid convolution multiplications by predicting and skipping near-zero valued data, given certain thresholds. Chen et al. [6] predict future weight updates during training based

on Momentum [35] parameters. Zhu et al. [44] use a fully connected layer to predict each ofm activation sign and low-rank approximation to decrease the weight matrix.

Spatial correlation. The correlation between neighboring activations in CNN feature maps is an inherent CNN characteristic that may be exploited. The works by Shomron and Weiser [38], Kim et al. [21], and Figurnov et al. [9] were described at Section 4.4. In addition, Kligvasser et al. [23] propose nonlinear activations with learnable spatial connection to enable the network capture more complex features, and Mahmoud et al. [32] operate on reduced precision deltas between adjacent activations rather than on true values.

Dynamic pruning. As opposed to static pruning [12][17][31][31][28], dynamic pruning is input-dependent. Dong et al. [7] present a network structure with embedded low-cost convolution layers that act as zero-valued activation masks for the original convolution masks. In a higher granularity, Lin et al. [29] use reinforcement learning for channel pruning, Gao et al. [10] prune channels according to a channel saliency map followed by a fully connected layer, and He et al. [14] propose a two-step channel pruning algorithm with LASSO regression and fine-tuning. All these works, except [14], involve extensive model training.

Our work is most closely related to the work of Dong et al., Shomron and Weiser, Kim et al., and Figurnov et al., with which we have also compared our results. However, our work provides the user with a continuous range of operating points and offers a prior estimate of the model accuracy degradation and MAC savings. Specifically, in contrast to Dong et al., our approach does not require labeled data (yet, it can be used for fine-tuning) and does not require hundreds of epochs for training. As for Shomron and Weiser, Kim et al., and Figurnov et al., we create the binary prediction masks using a CNN-based approach.

6 Conclusions

We propose a zero activation predictor (ZAP) that dynamically identifies the zero-valued output feature map (ofm) activations prior to their computation, thereby saving their convolution operations. ZAP exploits the spatial correlation of ofm activations inherent in convolution neural networks, meaning that according to a sparsely computed ofm, ZAP determines whether the remaining activations are zero-valued or non-zero-valued. ZAP is a lightweight CNN that imposes negligible computation and parameter overheads and its deployment and training does not require labeled data or modification of the baseline model architecture and parameters. In addition, ZAP speculation level is tunable, allowing an efficient *a priori* control of its accuracy-savings trade-off.

Acknowledgments We acknowledge the support of NVIDIA Corporation for its donation of a Titan V GPU used for this research.

References

1. Akhlaghi, V., Yazdanbakhsh, A., Samadi, K., Gupta, R.K., Esmaeilzadeh, H.: SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks. In: International Symposium on Computer Architecture (ISCA). pp. 662–673. IEEE (2018)
2. Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N.E., Moshovos, A.: Cnvlutin: Ineffectual-neuron-free deep neural network computing. In: International Symposium on Computer Architecture (ISCA). pp. 1–13. IEEE (2016)
3. Asadikouhanjani, M., Ko, S.B.: A novel architecture for early detection of negative output features in deep neural network accelerators. Transactions on Circuits and Systems II: Express Briefs (2020)
4. Canziani, A., Paszke, A., Culurciello, E.: An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678 (2016)
5. Chang, J., Choi, Y., Lee, T., Cho, J.: Reducing MAC operation in convolutional neural network with sign prediction. In: International Conference on Information and Communication Technology Convergence (ICTC). pp. 177–182. IEEE (2018)
6. Chen, C.C., Yang, C.L., Cheng, H.Y.: Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. arXiv preprint arXiv:1809.02839 (2018)
7. Dong, X., Huang, J., Yang, Y., Yan, S.: More is less: A more complicated network with less inference complexity. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5840–5848. IEEE (2017)
8. Elthakeb, A.T., Pilligundla, P., Esmaeilzadeh, H.: Divide and conquer: Leveraging intermediate feature representations for quantized training of neural networks. International Conference on Machine Learning (ICML) Workshop on Understanding and Improving Generalization in Deep Learning (2019)
9. Figurnov, M., Ibraimova, A., Vetrov, D.P., Kohli, P.: PerforatedCNNs: Acceleration through elimination of redundant convolutions. In: Advances in Neural Information Processing Systems (NIPS). pp. 947–955 (2016)
10. Gao, X., Zhao, Y., Dudziak, L., Mullins, R., Xu, C.z.: Dynamic channel pruning: Feature boosting and suppression. International Conference on Learning Representations (ICLR) (2018)
11. Geifman, Y., El-Yaniv, R.: Selective classification for deep neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 4878–4887 (2017)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778. IEEE (2016)
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: International Conference on Computer Vision (ICCV). pp. 1389–1397. IEEE (2017)
15. Hennessy, J.L., Patterson, D.A.: Computer architecture: a quantitative approach. Elsevier (2011)
16. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)

17. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
18. Hua, W., Zhou, Y., De Sa, C., Zhang, Z., Suh, G.E.: Boosting the performance of CNN accelerators with dynamic fine-grained channel gating. In: International Symposium on Microarchitecture (MICRO). pp. 139–150. ACM (2019)
19. Huan, Y., Qin, Y., You, Y., Zheng, L., Zou, Z.: A multiplication reduction technique with near-zero approximation for embedded learning in IoT devices. In: International System-on-Chip Conference (SOCC). pp. 102–107. IEEE (2016)
20. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. International Conference on Machine Learning (ICML) (2015)
21. Kim, C., Shin, D., Kim, B., Park, J.: Mosaic-CNN: A combined two-step zero prediction approach to trade off accuracy and computation energy in convolutional neural networks. Journal on Emerging and Selected Topics in Circuits and Systems **8**(4), 770–781 (2018)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR) (2014)
23. Kligvasser, I., Rott Shaham, T., Michaeli, T.: xUnit: Learning a spatial activation function for efficient image restoration. In: Conference on Computer Vision and Pattern Recognition (ECCV). pp. 2433–2442 (2018)
24. Krizhevsky, A., Hinton, G.: Convolutional deep belief networks on CIFAR-10. Unpublished manuscript **40**(7), 1–9 (2010)
25. Krizhevsky, A., Nair, V., Hinton, G.: CIFAR-10 and CIFAR-100 datasets. http://www.cs.toronto.edu/ kriz/cifar.html
26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 1097–1105 (2012)
27. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. Journal of Machine Learning Research **17**(1), 1334–1373 (2016)
28. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. In: International Conference on Learning Representations (ICLR) (2017)
29. Lin, J., Rao, Y., Lu, J., Zhou, J.: Runtime neural pruning. In: Advances in Neural Information Processing Systems (NIPS). pp. 2181–2191 (2017)
30. Lin, Y., Sakr, C., Kim, Y., Shanbhag, N.: PredictiveNet: An energy-efficient convolutional neural network via zero prediction. In: International Symposium on Circuits and Systems (ISCAS). pp. 1–4. IEEE (2017)
31. Luo, J.H., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: International Conference on Computer Vision (ICCV). pp. 5058–5066. IEEE (2017)
32. Mahmoud, M., Siu, K., Moshovos, A.: Diffy: a déjà vu-free differential deep neural network accelerator. In: International Symposium on Microarchitecture (MICRO). pp. 134–147. IEEE (2018)
33. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning (ICML). pp. 807–814 (2010)
34. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W., Dally, W.J.: SCNN: An accelerator for compressed-sparse convolutional neural networks. In: International Symposium on Computer Architecture (ISCA). pp. 27–40. IEEE (2017)

35. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Networks **12**(1), 145–151 (1999)
36. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015)
37. Sainath, T.N., Mohamed, A.r., Kingsbury, B., Ramabhadran, B.: Deep convolutional neural networks for LVCSR. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 8614–8618. IEEE (2013)
38. Shomron, G., Weiser, U.: Spatial correlation and value prediction in convolutional neural networks. Computer Architecture Letters (CAL) **18**(1), 10–13 (2019)
39. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
40. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. International Conference on Machine Learning (ICML) (2015)
41. Song, M., Zhao, J., Hu, Y., Zhang, J., Li, T.: Prediction based execution on deep neural networks. In: International Symposium on Computer Architecture (ISCA). pp. 752–763. IEEE (2018)
42. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)
43. Yazdani, R., Riera, M., Arnau, J.M., González, A.: The dark side of DNN pruning. In: International Symposium on Computer Architecture (ISCA). pp. 790–801. IEEE (2018)
44. Zhu, J., Jiang, J., Chen, X., Tsui, C.Y.: Sparsenn: An energy-efficient neural network accelerator exploiting input and output sparsity. In: Design, Automation and Test in Europe Conference (DATE). pp. 241–244. IEEE (2018)