Activation Pruning of Deep Convolutional Neural Networks

Arash Ardakani, Carlo Condo and Warren J. Gross
Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada

***Abstract*—Deep neural networks (DNNs) have risen to prominence, in the last few years, thanks to their very good performance on different classification and recognition tasks. However, their implementations suffer from long latency caused by the complexity of the network. Recently, many hardware implementations were introduced to accelerate the processing time of DNNs, and in particular of convolutional layers. While they can easily meet the timing constraint of real-time applications for small networks, complex models such as VGG and VGG-like networks are still out of reach. In this paper, we propose an technique to prune the neurons of each convolutional layer, also called activations, which directly contribute to the latency. Comparing networks with the same number of activations, we show that the activation-pruned networks perform better than the unpruned networks in terms of misclassification error.**

I. Introduction

Deep neural networks (DNNs) have exhibited excellent performance in many complex tasks, such as classification and recognition tasks [1]. Due to their remarkable capability in extracting features and processing raw data, they have been adopted in many real-time applications. However, utilization of these networks in such applications requires high-performance platforms such as graphics processing units (GPUs) and central processing units (CPUs), due to the complexity and size of their models.

In the past few years, the rapid growth of devices with sensing and processing capabilities connected to the internet lead to the concept of Internet of Things (IoT). IoT devices are based on a variety of hardware platforms and networks that change dynamically over time [2]. To meet the demands of heterogeneity and dynamic changes within the IoT framework, machine learning algorithms are one of the most promising solutions [2]. However, state-of-the-art DDN models have a high degree of complexity and are over parametrized, resulting in a higher latency and power/energy consumption than the budget of IoT devices.

To address this issue, general purpose processors have been replaced by application-specific integrated circuits (ASICs) for hardware implementations of DNNs, since custom hardware consumes less power and results in lower latency compared to GPUs and CPUs [3]. While custom hardware significantly reduces the energy/power consumption as well as the latency of the system, they still cannot meet the constraints of real-time systems as the size of model increases. For example, the computations of convolutional processes of VGGNet-16 [4] require few seconds due to their high complexity [5].

Recently, many approaches were introduced to solve this problem. Spiking neural networks and stochastic models [6]–[10] strove to exploit parallel paradigms where numbers are represented as sequences of random bits. They allow to perform multiplications through simple xnor gates, which significantly limit the power/energy consumptions. However, substantial power is dissipated through the required parallel access to the off-chip memory. Moreover, due to the nature of stochastic computing, stochastic systems still result in long latencies.

Another approach which has been considered by numerous works is the binarization/ternarization of networks [11], [12]. In this method, the weights are represented using binary/ternary values, that allow to perform multiplications using multiplexers. Although this method improves the efficiency of the system in terms of power/energy consumption by reducing the required bit-width of input data, the number of parameters and activations remains unchanged.

Finally, pruning methods were introduced mainly to reduce the number of parameters of DNNs [13], [14]. Pruning techniques effectively reduce the number of memory accesses to the off-chip memory, that dominates the power consumption of DNNs, while they can result in 3× to 4× layer-wise speedup for fully-connected layers. However, no practical speedup were obtained for convolutional layers on custom hardware [15].

In this paper, we propose an activation-pruning method to reduce the number of activations of each convolutional layers, that directly contributes to the latency of the system. We then show that hardware-friendly linear-feedback shift registers (LFSRs) can be exploited to efficiently identify the existing activations. Finally, we compare the pruned network to an unpruned network with the same number of activations: we show that the pruned network performs better than its unpruned counterpart in terms of misclassification rate.

II. Preliminaries

A. Deep Neural Network

DNNs are stacks of multiple layers of neurons, also called activations, between the input and output layers. Through an initial training stage, the weights associated to each connection among neurons are adjusted. Afterwards, the extracted weights are used to classify, recognize and process data in the inference stage. One of the most important subcategories of DNNs is that of convolutional neural networks (CNNs), widely used in recognition and detection tasks. CNNs are stacks of convolutional layers followed by a few fully-connected layers. The

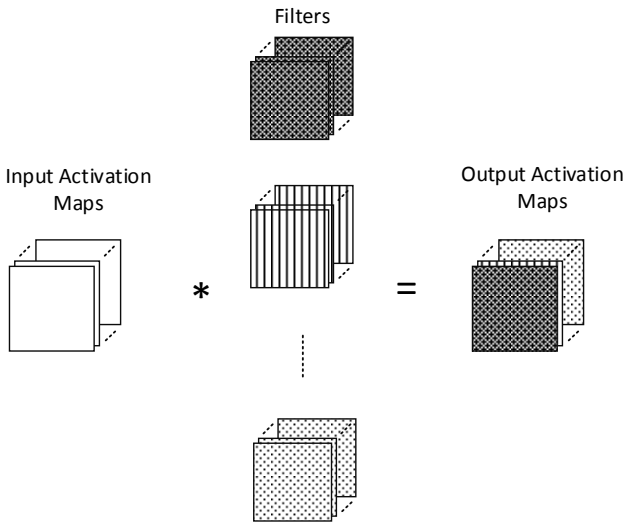

Figure 1. Shows a high-dimensional convolution.

connectivity between neurons in convolutional layers follows a pattern inspired by the organization of the visual cortex of animals, that can mathematically described by convolution operations [16]. Therefore, only few neurons are connected together and all the neurons share a set of weights. The main objective of convolutional layers is to extract high level abstractions and features of data. In contrast, fully-connected layers learn non-linear combinations of data.

DNNs are trained by computing the gradient of a cost function C with respect to the weights of all layers using the backpropagation algorithm, in conjunction with stochastic gradient descent (SGD) optimization method [17]. This technique feeds the DNN a set of training data and backward propagates the obtained errors through all the layers to update the weights, by minimizing the cost function. Among all the cost models reported in literature, the modified hinge loss is a common choice for the cost function due to its remarkable performance [18]. The input data is first divided in mini-batches and network parameters are updated using each mini-batch several times. This technique reduces the convergence time of the training algorithm. Batch normalization has become a common approach to regularize each mini-batch of data. It allows the use of a bigger η to speed up the training process, where η is a learning rate used to control the weight updating speed.

B. Convolutional Computations

A convolutional layer consists of activations arranged in 3 dimensions, 4 dimensional filters (each composed of a set of 3 dimensions filters) and 3 dimensional output activations, performing high-dimensional convolutions as shown in Fig. 1. Each single 2D plane, referred to as a channel, of the output activation maps is a result of the convolution between 3D input activation maps and a set of 3D filters. Therefore, a summation of multiple plane-wise 2D convolutions forms a 3D convolution. It is worth noting that a 1D bias is also added to the 3D convolution results.

During the past few years, custom hardware architectures were proposed to accelerate convolutional computations [5], [19]. While they managed to substantially reduce the power/energy consumption as well as latency of convolutional layers, they failed to meet the timing constraint of the real-time systems as the model complexity increases. On the other hand, the majority of current research efforts tend towards a better accuracy: this leads to very complex models, for which real-time computations, even with custom hardware, have not been achieved.

Recently, many pruning techniques have been introduced to highly reduce the number of parameters of DNNs [13]–[15]. While they manage to significantly reduce the number of power-hungry memory accesses, no practical speed up has reported on custom hardware in literature so far. It is worth mentioning that some works such as [15] managed to obtain 5.1× speedups for convolutional processes on GPUs and CPUs, however, the power consumption of these platforms is significantly higher than the power budget of IoT systems.

III. Proposed Activation Pruning Method

The main computational elements of convolutional layers that directly affect the latency of the system are the number of filter kernels and number of output activations. Filter kernels are 3D filters composed of multiple 2D matrices. A common trend in the research community has seen the increment in the number of filter kernels to obtain a better accuracy. The number of 2D planes of the output activation maps are determined by the total number of 3D filters. Therefore, while decreasing the number filter kernels reduces the complexity and consequently the latency of the system, it also greatly reduces its accuracy.

In this paper, we propose to prune the output activations of convolutional layers while using the same number of filter kernels as the unpruned models, leading to a lower latency. To this end, let us consider the forward computations a convolutional layer as follows:

$$y(t) = act(X * W(t) + b(t)), \tag{1}$$

where y represents the 3D output activation maps, X the 3D input activation maps, W the 4D weights and b 1D biases. Also, $*$ denotes the 3D convolutional process. $\mathrm{ReLU}(x) = \max(0, x)$ is used in most cases as the non-linear activation function $act()$.

Randomly dropping connections has shown a remarkable performance when used in fully-connected layers [14]. In [14], it was shown that the connections of the fully-connected layers can be randomly reduced while improving accuracy of DNNs. Similarly, in this paper, we propose to randomly form the mask matrix m using binary values to randomly disable some output activations. Thus, the sparse activation matrix y_s can be defined as

$$y_s = y \cdot m, \tag{2}$$

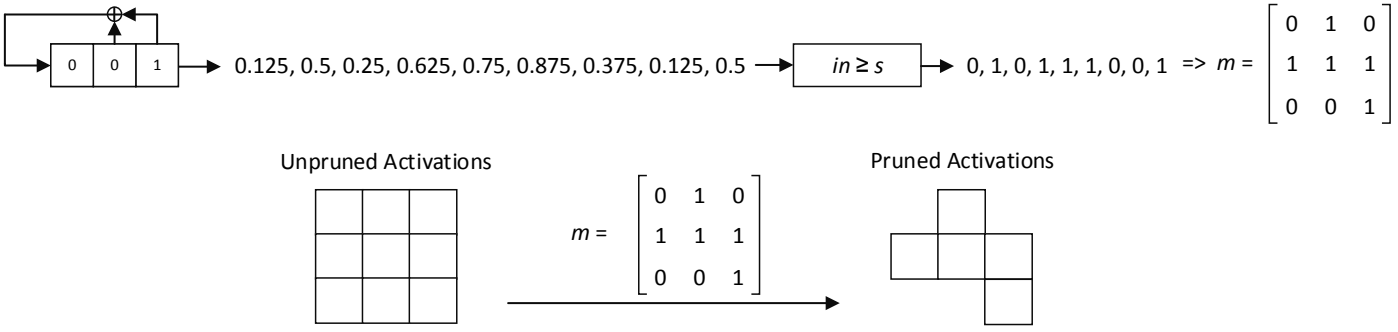

Figure 2. Shows the proposed activation pruning method using an SNG.

Algorithm 1: Training algorithm for the proposed activation pruning method

Data: Convolutional layers with parameters W, b and m for each layer. Input data X, its corresponding targets Y, and learning rate of η.

Result: W and b

1. Forward computations

for *each layer i in range(1,N)* **do**

  Compute temporary layer output y_t according to (1) and its previous layer output y_{i-1}, W and b_i.

  $y_s \leftarrow y_t \cdot m_i$

  $y_i \leftarrow y_s$

end

2. Backward Computations

Initialize output layers activation gradient $\frac{\partial C}{\partial y_N}$

for *each layer j in range(2,N-1)* **do**

  Compute $\frac{\partial C}{\partial y_j}$

end

for *each layer j in range(1,N-1)* **do**

  Compute $\frac{\partial C}{\partial W_s}$ knowing $\frac{\partial C}{\partial y_j}$ and y_{j-1}

  Compute $\frac{\partial C}{\partial b_j}$

  Update $W_j : W_j \leftarrow W_j - \eta \frac{\partial C}{\partial W}$

  Update $b_j : b_j \leftarrow b_j - \eta \frac{\partial C}{\partial b_j}$

end

where $(\cdot)$ denotes the element-wise multiplication and the size of m is the same as the size of y. It is also worth noting that y consists of several 2D activation planes. The sparse activation model can be trained using the SGD algorithm, as summarized in Algorithm 1.

In order to make the proposed technique hardware friendly, we propose the use of LFSRs to form each plane of output activation maps, similarly to the approach used in [14]. In this way, each output activation plane is associated with an n-bit LFSR serially generating $2^n - 1$ numbers $S_i \in (0, 1)$, $i \in \{1, 2, \ldots, 2^n - 1\}$. Comparing S_i with a constant value s outputs a random binary stream with expected value of $s \in [0, 1]$. This unit, which is widely used in stochastic computing, is referred to as stochastic number generator (SNG). Each of the SNG outputs X_i is either 1 or 0, depending on the value of $S_i \geq s$. Therefore, the sparsity degree of the activations is tunable by changing the value of s. Fig. 2 shows the proposed pruning procedure on a single 2D plane of the output activation maps while using an SNG.

IV. Experimental Results

We used the Theano library in Python [20] to validate the effectiveness of the proposed activation pruning method on the CIFAR10 dataset. This dataset consists of a total number of 60000 32×32 color images, 50000 for training and 10000 for testing, falling into 10 classes. We divide the training images into 40000 and 10000 training and validation datasets. As out model, we adopt a convolutional neural network inspired by VGGNet, also used in [11]. This model comprises $\{128-128-256-256-512-512\}$ channels for six convolutional/pooling layers and two 1024-node fully-connected layers followed by a classifier. Hinge loss and batch normalization are also used in our training while using a batch size of 50.

Recently, binary and ternary networks have shown a remarkable performance on different datasets [11], [12] in which the weights are represented with either -1 or 1 in binary networks, and -1, 0, or 1 in ternary networks. These networks significantly reduce the bit-width of the weights and remove multiplications, making them suitable for hardware implementations. Moreover, in [14], it was shown that networks trained by binary or ternary weights perform better than the single precision floating point models. Therefore, we also perform our simulations using ternary weights.

Table I summarizes the misclassification rates of the proposed activation pruning method on CIFAR10 dataset. In our simulations, we used the VGG-like network as a baseline and pruned the output activations of each convolutional layer of the baseline for different degrees of sparsity. In order to show the

Table I
MISCLASSIFICATION RATE (MCR) FOR DIFFERENT DEGREE OF SPARSITY ON CIFAR10 DATASET

Channel Configuration of Convolutional Layers	Sparsity Degree of Activation	MCR (%) [CIFAR10]	Total Number of Activations	Channel Configuration of Convolutional Layers	Sparsity Degree of Activation	MCR (%) [CIFAR10]	Total Number of Activations
{128 – 128 – 256 – 256 – 512 – 512}	0% (Baseline)	9.32	458752	–	–	–	–
{128 – 128 – 256 – 256 – 512 – 512}	20% (Proposed)	9.99	367002	{103 – 103 – 205 – 205 – 410 – 410}	0% (Conventional)	10.25	368384
{128 – 128 – 256 – 256 – 512 – 512}	30% (Proposed)	12.12	321126	{90 – 90 – 180 – 180 – 359 – 359}	0% (Conventional)	13.06	322432
{128 – 128 – 256 – 256 – 512 – 512}	40% (Proposed)	12.4	275251	{77 – 77 – 154 – 154 – 308 – 308}	0% (Conventional)	13.41	275968
{128 – 128 – 256 – 256 – 512 – 512}	50% (Proposed)	12.47	229376	{64 – 64 – 128 – 128 – 256 – 256}	0% (Conventional)	14.52	229376
{128 – 128 – 256 – 256 – 512 – 512}	60% (Proposed)	13.67	183501	{52 – 52 – 104 – 104 – 205 – 205}	0% (Conventional)	14.82	185984
{128 – 128 – 256 – 256 – 512 – 512}	70% (Proposed)	15.48	137626	{39 – 39 – 77 – 77 – 154 – 154}	0% (Conventional)	15.33	139008
{128 – 128 – 256 – 256 – 512 – 512}	80% (Proposed)	17.29	91750	{26 – 26 – 52 – 52 – 104 – 104}	0% (Conventional)	16.38	93184
{128 – 128 – 256 – 256 – 512 – 512}	90% (Proposed)	18.96	45875	{13 – 13 – 26 – 26 – 52 – 52}	0% (Conventional)	19.55	46592

effectiveness of the proposed activation pruning method over the unpruned networks, we compare to unpruned networks that contain the same number of output activations. These are obtained by reducing the number of output activation planes (i.e. channels of each convolutional layer). Simulation results show that the misclassification rate increases as the sparsity degree of the proposed activation pruning method increases. However, the pruned networks perform better than the unpruned networks in the majority of cases, with a peak of 2.05%. We used hardware friendly SNGs to denote the existence of the output activations. Therefore, only the seed values of LFSRs are required to be stored in a memory for hardware implementation. The proposed method can also be used to speed up the convolutional computations. For instance, considering the target misclassification rate of 13.5%, an unpruned network requires $275,968$ activations, while the proposed pruned network only requires $183,501$ activations to be computed. Depending on the CNN architecture, it can results in up to 33% lower latency.

V. CONCLUSION AND FUTURE WORKS

In this paper, we proposed an activation pruning method for convolutional layers of CNNs. We showed that the proposed method, when comparing networks with the same resulting number of output activations, performs better than the conventional unpruned networks on the CIFAR10 dataset in terms of misclassification rate, peaking at a 2.05% improvement. This approach reduces the latency and the memory accesses to the off-chip memory, which makes it suitable to be exploited in real-time applications. In future works, we plan to test the proposed method on other datasets and models. Moreover, we will test the proposed method on a custom hardware to report measured numbers for the power and latency.

REFERENCES

[1] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015.

[2] Ovidiu Vermesan and Peter Friess, *Internet of things-from research and innovation to market deployment*, River Publishers Aalborg, 2014.

[3] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 243–254.

[4] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.

[6] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–12, 2017.

[7] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth: design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[8] Sean C. Smithson and Kaushik Boga and Arash Ardakani and Brett H. Meyer and Warren J. Gross, "Stochastic computing can improve upon digital spiking neural networks," in *2016 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2016, pp. 309–314.

[9] Ji Li, Zihao Yuan, Zhe Li, Caiwen Ding, Ao Ren, Qinru Qiu, Jeffrey Draper, and Yanzhi Wang, "Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks," *arXiv preprint arXiv:1703.04135*, 2017.

[10] Vincent T Lee, Armin Alaghi, John P Hayes, Visvesh Sathe, and Luis Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 13–18.

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

[12] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio, "Neural Networks with Few Multiplications," *CoRR*, vol. abs/1510.03009, 2015.

[13] Song Han, Jeff Pool, John Tran, and William J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," *CoRR*, vol. abs/1506.02626, 2015.

[14] Arash Ardakani, Carlo Condo, and Warren J Gross, "Sparsely-Connected Neural Networks: Towards Efficient VLSI Implementation of Deep Neural Networks," *Accepted for publication in the 5th International Conference on Learning Representations (ICLR)*, 2016.

[15] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li, "Learning Structured Sparsity in Deep Neural Networks," *CoRR*, vol. abs/1608.03665, 2016.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1," chapter Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986.

[18] Yichuan Tang, "Deep Learning using Support Vector Machines," *CoRR*, vol. abs/1306.0239, 2013.

[19] B. Moons and M. Verhelst, "An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, April 2017.

[20] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.