Accelerating Neural Networks using Channel Gating

I. MOTIVATION

Deploying convolutional neural networks (CNNs) effectively in real-time applications often requires both high throughput and low power consumption. However, a state-of-the-art CNN typically performs about 10^9 FLOPs per evaluation [1]. Reducing this computational cost has become an essential challenge. Several prior studies have proposed *pruning* ineffectual features and weights statically, thus reducing the FLOPs [2]. Dedicated hardware accelerators have also been shown to improve performance by exploiting the sparsity in a CNN [3], [4], [5]. However, the aforementioned proposals suffer from increasing the irregularity of the computation at the finest granularity.

We propose a novel approach to reduce CNN computation, called *channel gating*, which *dynamically* prunes the unnecessary computation specific to a particular image, while minimizing the accuracy loss and hardware modification. Intuitively, channel gating leverages the spatial information inside the input features to identify ineffective receptive fields and skip the corresponding computation by gating a fraction of the input channels.

The paper makes the following major contributions:

- We introduce the channel gating scheme, which dynamically prunes computation on input channels at receptive field level.
- We propose an efficient single-pass training scheme to train the channel gating CNN from scratch, allowing the network to automatically learn an effective gating policy.
- We demonstrate the benefits of introducing channel gating in CNNs empirically and get 66% and 60% reduction in FLOPs with 0.22% and 0.29% accuracy loss on the CIFAR-10/100 datasets respectively using a state-of-the-art ResNet model.
- We propose a specialized accelerator architecture, which improves the performance and energy efficiency of the channel gating CNN inference **(Ongoing)**.

II. METHODOLOGY

A. Channel gating neural network

We introduce channel gating using a single neuron example. Figure 1 shows a neuron taking in c input channels. The c channels split into two groups, where one group contains the p channels and the other has the remaining r channels. By comparing the dot product ($\sum_{i=1}^{p} w_i x_i$) with a learnable threshold, the gate turns the r channels on or off. Assuming ReLU activation is used, the gate function (s) is defined using the step function.

$$\theta(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

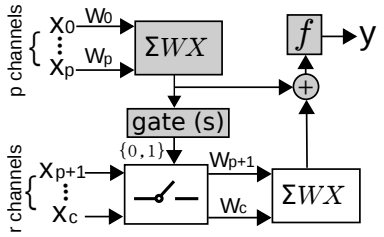


Fig. 1: Channel gating on a single neuron.

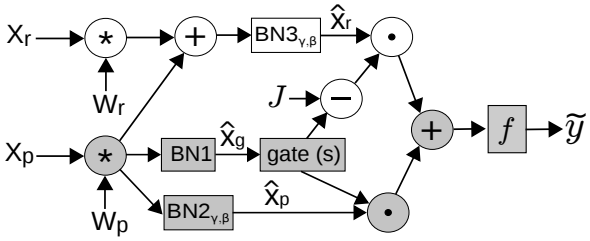


Fig. 2: Computational graph of the channel gating building block.

$$\mathrm{s}(\mathbf{x}, \Delta) = \theta(\mathbf{x} - \Delta) \tag{2}$$

In practice, the channel gating scheme is applied to any k by k convolutional and fully-connected layers where k is the width of the filter. A binary decision is generated for each output pixel where a binary value **1** in decision tensor ($\mathbf{d}_{i,j,k}$) stands for skipping $k^2 r$ FLOPs in computing the output pixel $y_{(i,j,k)}$.

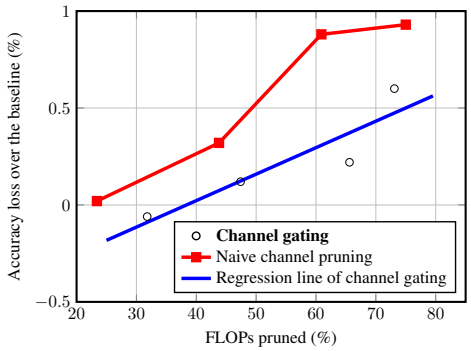
B. Training a network with channel gating

The channel gating scheme prevents us from utilizing the standard training method in two aspects: **(1)** introduce a non-differentiable gate function; **(2)** need to encourage the network to reduce computation.

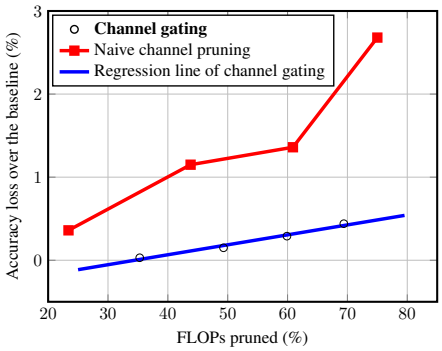
Non-differentiable gate function. As shown in Fig. 2, we construct the computational graph of the channel gating block. The gradient towards $\hat{\mathbf{x}}_\mathbf{g}$ and Δ cannot be computed directly since the gate is a non-differentiable function. We approximate the gate with a smooth function only during backward propagation.

$$\mathbf{s}(\mathbf{x}, \Delta) = \frac{1}{1 + e^{\epsilon \cdot (\mathbf{x} - \Delta)}} \tag{3}$$

Pruning strategy. The FLOPs reduction fraction (F) is a function of the threshold (Δ) and F increases monotonically with Δ. As a result, reducing computation cost is equivalent to having a larger Δ. To minimize the computational cost during



(a) CIFAR-10



(b) CIFAR-100

Fig. 3: Accuracy vs. FLOPs reduction trade-offs with different approaches.

the training, we set a target threshold value (T) for all the layers and add the squared loss of the difference between Δ and T ($|T - \Delta|^2$) into the loss function.

C. Hardware architecture

We propose three architectural changes to accelerate the channel gating networks. First, only a small fraction of the receptive fields (k by k windows) in the $W \times H$ plane of x_r are needed because of channel gating. Thus, x_r and w_r is stored along the channel dimension to maximize the data locality. Based on the run-time decisions, a subset of vectors in x_r are fetched into on-chip buffer where the width and height indices are used as the index of each vector. Secondly, we propose to accelerate $w_r \cdot x_r$ with vector-vector multiplier other than systolic array since both inputs are vectors with r elements. Lastly, given that no weight-sharing in the fully-connected layers, conditional weight fetching are implemented to minimize the communication overhead.

III. RESULT

As shown in Figure 3, we demonstrate that the channel gating scheme outperforms a naive approach which prunes a fixed fraction of the channels statically and trains the pruned model from scratch. In Table I, we summarize the best channel gating models in terms of the accuracy and FLOPs reduction trade-off.

TABLE I: The best test error and FLOPs trade-off points on CIFAR-10 and CIFAR-100.

Dataset	Model	Test Error (%)	FLOPs	Pruned (%)
CIFAR-10	Res-18-baseline	5.40	5.01×10^8	
	Res-18-pruned-A	5.34	3.42×10^8	31.8
	Res-18-pruned-B	5.52	2.64×10^8	47.4
	Res-18-pruned-C	5.62	1.72×10^8	65.6
	Res-18-pruned-D	6.00	1.35×10^8	73.1
CIFAR-100	Res-18-baseline	24.95	5.01×10^8	
	Res-18-pruned-A	24.98	3.24×10^8	35.3
	Res-18-pruned-B	25.10	2.54×10^8	49.3
	Res-18-pruned-C	25.24	2.00×10^8	59.9
	Res-18-pruned-D	25.39	1.53×10^8	69.4

REFERENCES

[1] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678, 2016. [Online]. Available: http://arxiv.org/abs/1605.07678

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[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," *SIGARCH Comput. Archit. News*, vol. 44, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/3007787.3001163

[4] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *SIGARCH Comput. Archit. News*, vol. 44, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/3007787.3001138

[5] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.