Late Breaking Results:

LODESTAR: Creating Locally-Dense CNNs for Efficient Inference on Systolic Arrays*

Bahar Asgari
Georgia Tech

Ramyad Hadidi
Georgia Tech

Hyesoon Kim
Georgia Tech

Sudhakar Yalamanchili
Georgia Tech

ABSTRACT

The performance of sparse problems suffers from lack of spatial locality and low memory bandwidth utilization. However, the distribution of non-zero values in the data structures of a class of sparse problems, such as matrix operations in neural networks, is modifiable so that it can be matched with an efficient underlying hardware, such as systolic arrays. Such modification helps addressing the challenges coupled with sparsity. To efficiently execute sparse neural network inference on systolic arrays, we propose a structured pruning algorithm that increases the spatial locality in neural network models, while maintaining the accuracy of inference.

ACM Reference Format:
Bahar Asgari, Ramyad Hadidi, Hyesoon Kim, and Sudhakar Yalamanchili. 2019. Late Breaking Results: LODESTAR: Creating Locally-Dense CNNs for Efficient Inference on Systolic Arrays. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3316781.3322472

1 INTRODUCTION AND MOTIVATION

Systolic arrays [7] have seen a resurgence for implementing in convolutional neural networks (CNNs) inference, a practical example of which is in Google's TPU [6]. Systolic arrays eliminate the need for irregular intermediate accesses to the memory hierarchy, and capture data reuse patterns. This approach works particularly well for computing linear recurrences and *dense* linear algebra computations. However, inference using CNNs is a *sparse* problem, which presents significant efficiency challenges such as underutilization of memory bandwidth due to storing data in sparse formats, indirect memory accesses and transferring of extra meta data.

CNN inference is sparse because, during training, several of weights are assigned close-to-zero values. Thus, to reduce the amount of computation as well as the memory footprint, the close-to-zero values are usually pruned. Since pruning the individual values of a model [5, 9] results irregular models with consequences of resource underutilization and high storage overhead, *structured* pruning techniques have been proposed, which prune the weights at the granularity of a vector [1, 9], kernel [1, 9], filter [8–11], channel [1, 11], or entire layer [11], all of which are optimizations for CPUs and GPUs and help in reducing the number of operations, memory footprint, and computation complexity.

The main challenge is that the preceding optimizations are insufficient to exploit the data reuse in systolic arrays, and the highly concurrent, synchronous, and rhythmic flow of data from memory. In fact, the storage adjacency of data resulting from algorithm-defined pruning (e.g., kernel, filter) is not necessarily matched with data organizations necessary to directly stream to the interacting data flows in the systolic array. Hence, our goal is to propose a pruning, the output of which is compatible with a streaming memory interface to eliminate external buffering/caching for compute.

2 PROPOSED APPROACH

We propose creating locally-dense CNNs for efficient inference on systolic arrays (LODESTAR), to enable streaming of sparse data from memory to exploit the distinctive *data reuse* patterns and fine-grained concurrency of systolic arrays. LODESTAR produces a *weight matrix* such that the non-zero values are clustered spatially into *locally-dense* regions, which are compactly stored and efficiently *streamed*. We examine the correlation among all the filters, which differs from pruning the individual filters of a CNN [8–11]. To sustain accuracy, we may keep more number of non-zeros compared to common pruning algorithms. However, in achieving higher performance and efficiency, the distribution of non-zeros is more influential than their quantity, when optimized for streaming data.

Efficiently using systolic arrays for matrix multiplication is applicable to CNNs by converting their convolutional operations to general matrix-matrix multiplication (GEMM), and flattening 4D weight matrices to 2D ones. Besides the known benefits of using GEMM for CNNs [3, 4], it offers the opportunity for creating a locally-dense data by considering correlated filters together. We prune the flattened weight matrix (i.e., $W_{K \times F^2C}$, K:#filters, C:#channels, F:filter size) to extract non-zero blocks, the width of which are selected to match with the width of the target systolic array, ω. The matched widths of the non-zero blocks and that of the systolic array guarantees the correctness of multiplications. The non-zero blocks of the produced matrix are streamed into the systolic array. The blocks are created by splitting large weight matrices into F^2C/ω chunks, and extracting the non-zero blocks in

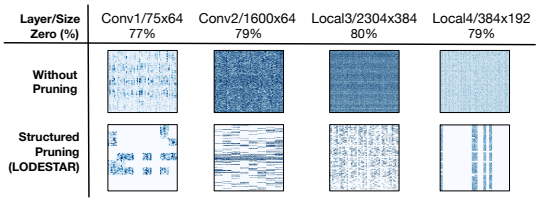

Figure 1: Applying Algorithm 1 with $l, \omega = 8$ on CifarNet. The accuracy after pruning 79.8% of model is 93.6%.

*Supported by NSF CCF 1533767. All correspondence to bahar.asgari@gatech.edu.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3322472

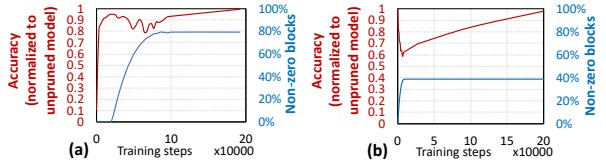

Figure 2: Accuracy and the percentage of zero blocks: a) CifarNet (pruning between step 20k and 100k), and b) VGG16 (pruning between steps 1 and 10k).

each chuck. The adjacent blocks of the pruned models are concatenated and stored by assigning them a single index (i.e., the column index of the first block) and a single length.

Unlike common pruning algorithms, Algorithm 1 (i) incurs some increase in sparsity while producing locally-dense data, and (ii) applies pruning to GEMM flattened operand and not individual weights. The input parameters of the pruning algorithm are the weight matrix W, threshold θ, length of the window l, and width of the systolic array ω. The width of the window is fixed and is equal to ω. The weight matrix is either the flattened version of the weight matrix in a convolution layer, or the 2D weight matrix itself in a fully-connected layer. During pruning, a window of size ωxl slides over W. The size l is a hyperparameter. We choose l=8, which offers the best trade-off between sparsity of blocks and storage overhead.

Algorithm 1 Pruning method of LODESTAR

```
1:  function PRUNE(W_{hxw}, θ, l, ω)
        W_{hxw}: Weight matrix,   θ: Threshold,
        ω: Systolic array width   l: Window length
2:      i_h := 0, i_w := 0, avg := 0
3:      while i_w < w do
4:          avg = BlockAvg([i_w, i_h], [i_w + ω − 1, i_h + l − 1])
5:          if avg < θ then
6:              W[i_w : i_w + ω − 1, i_h : i_h + l − 1] = 0
7:              i_h = i_h + l
8:          else
9:              i_h = i_h + 1
10:         if i_h > h − l then
11:             i_h = 0 , i_w = i_w + ω
```

During retraining, by increasing θ in later epochs, the algorithm maintains the accuracy and convergence. The windows are non-overlapping in x- and y-axes. Non-overlapping windows in x-axes is necessary to match with systolic-array width, and in y-axes for reducing the complexity of the problem from a global to local optimization. Figure 1 illustrates an example of applying the algorithm on CifarNet. Algorithm 1 does not change the size of the common axis of the operands of GEMM (i.e., F^2C), which leads to following benefits: (i) no need to change the dimensions of the input matrix (image), and (ii) both the pruned matrix (weights) and the dense matrix (inputs) can be either streamed through the systolic array or be the stationary operand during the multiplication. Thus, based on the size of the matrices at each layer of a CNN, we can dynamically swap the role of the two matrices to be streamed or stationary.

3 SIGNIFICANT FINDINGS

Methodology: For iteratively pruning and training three CNN models, VGG16, CifarNet, and LeNet on ImageNet, Cifar10, and MNIST datasets, we use Tensorflow™. To compare the inference performance of structured models with that of baselines, all executed on systolic arrays, we model a 64×64 systolic array (3 cycles latency @2GHz for multipliers and adder trees, similar to [2]) connected to high-bandwidth memory (HBM), using an in-house cycle-level simulator. The model estimates the power consumption by using

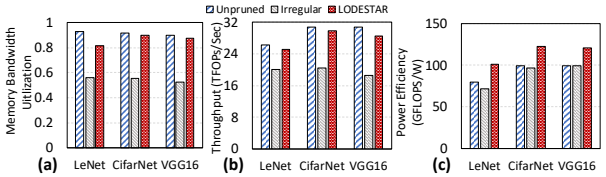

Figure 3: (a) Memory bandwidth utilization, (b) Throughput, and (c) Power efficiency of LODESTAR and the baselines.

Kitfox1.1 library at 16nm technology and McPAT model for compute units. We assume access energy of 6 pJ/bit for HBM.

Accuracy: Figure 2 illustrates the test accuracy (normalized to unpruned) and the percentage of zero blocks of CifarNet and VGG16 models, during training steps. For VGG16, we use a pre-trained model so we start pruning from the beginning. During pruning, as the percentage of zero blocks increases, the distribution of zero blocks and/or their densities keep changing, and the accuracy oscillates. After stoping pruning, training continues to maximize the accuracy. For LeNet, CifarNet, and VGG16, we prune 75%, 79.8%, and 40% of models and respectively achieve 99%, 93.6%, and 70% top-1 accuracy on validation set. The top-1 accuracy of unpruned models are 99% for LeNet, 94% for CifarNet, and 71.5% for VGG-16.

Performance: We compare the performance of LODESTAR against irregular sparse CNNs and unpruned models from three perspectives: *(i) Memory bandwidth utilization:* as Figure 3a illustrates, similar to the unpruned models, LODESTAR utilizes memory bandwidth better than irregular models. The reasons are streaming data, less number of memory references, and less meta-data for storing the sparse models; *(ii) Throughput:* in addition to bandwidth utilization, compute utilizations impacts the throughput. Figure 3b illustrates the effect of both on throughput. Although the number of operations in a structured model could be more than those in an irregular sparse model, the locality of them in leads lower latency. As a result, the combination of fast computation and high bandwidth utilization leads LODESTAR to work closer to the peak throughput of the engine, which is 32.78TFLOPs/Sec (i.e., 512GB/s $\times$ 64FLOP/B); *(iii) Power efficiency:* LODESTAR impacts the power consumption by reducing the number of memory accesses and by modifying the number of operations. In terms of the contribution of memory accesses, LODESTAR works better than irregular pruning. However, the contribution of computations in power consumption is the opposite, because the structured pruning may increase the number of operations, and the systolic array executes spatial operations much faster than sparse operation. Thus, the ratio of memory-access reduction to increase in compute density is a key factor. As Figure 3c shows, the reduction in memory accesses carries more weight and helps achieving better power efficiency.

REFERENCES

[1] Anwar, S., et al. Structured pruning of deep cnns. *JETC 13*, 3 (2017), 32.
[2] Asgari, B., et al. Memory slices: A modular building block for scalable, intelligent memory systems. *arXiv preprint arXiv:1803.06068* (2018).
[3] Chetlur, S., et al. cudnn: Efficient primitives for deep learning. *arXiv:1410.0759*.
[4] Hadjis, S., et al. Caffe con troll: Shallow ideas to speed up deep learning. In *Proceedings of the Fourth Workshop on Data analytics in the Cloud* (2015), ACM.
[5] Han, S., et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149* (2015).
[6] Jouppi, N. P., et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA* (2017), IEEE, pp. 1–12.
[7] Kung, H.-T. Why systolic architectures? *IEEE computer 15*, 1 (1982), 37–46.
[8] Li, H., et al. Pruning filters for efficient convnets. *arXiv:1608.08710* (2016).
[9] Mao, H., et al. Exploring the regularity of sparse structure in cnns. *arXiv:1705.08922*.
[10] Molchanov, P., et al. Pruning convolutional neural networks for resource efficient inference. *arXiv:1611.06440*.
[11] Wen, W., et al. Learning structured sparsity in dnns. In *NIPS* (2016).