# SparseCore: An Accelerator for Structurally Sparse CNNs

Extended Abstract

Sharad Chole
sharadchole@gmail.com

Ramteja Tadishetti
ramtejatadishetti@gmail.com

Sree Reddy
sreereddy@gmail.com

## ABSTRACT

Convolutional Neural Networks (CNN) have achieved super human accuracies in numerous machine learning tasks. Increasing model sizes have made them highly resource intensive and power hungry, making them unfit for embedded platforms such as wearables, hand-held devices, drones, autonomous robots etc. To address these limitations, we introduce a) A novel pruning scheme, ***k/m*** Sparsity, which enforces structured constraints among elements of weight tensors. b) An accelerator architecture called **SparseCore**, that achieves high energy efficiency and lowers the resource footprint by effectively exploiting *k/m* sparsity in weights and activations. On ILSVRC2012 dataset, *k/m* pruning scheme results in 50% to 75% sparsity without loss of accuracy. Our experiments show that SparseCore can achieve close to **90%** ALU utilization, improving both performance and energy efficiency by **3.1x** and **2.5x** respectively, compared to dense CNN inference.

## 1 INTRODUCTION

Convolutional Neural Networks have emerged as the foundation for applications such as speech recognition[3], image recognition[6], NLP[2] and ADAS[7]. Their ability to generalize and achieve greater accuracy comes at the cost of increased model complexity. While Graphics Processing Units (GPUs) are widely used to accelerate CNNs for datacenter training and inference, cost and power requirements for such systems make them prohibitively expensive for deployment for edge applications. This work aims to address the problem by proposing optimization of the model sparsity targeted towards an accelerator architecture.

### 1.1 Sparsity in CNNs

Sparsity refers to the existence of zeros in weight and input activation tensors. Recent publications have shown that many common networks can be pruned dramatically during training without loss of accuracy[5][4]. Such pruning can reduce the number of non-zero weights in convolution layers by 20% to 80%.

Dynamic sparsity in input activation arises due to activation function ReLU (Rectified Linear Unit). This non-linear operator clamps all negative values to zero, only allowing positive values to go through. This can result in as high as 70% of the activations being zero in convolution layers. These inherent sparse activations can be used to optimize processing by means of compression and hence reduction in overall ALU operations.

### 1.2 Hardware Accelerators

Various accelerator architectures have been proposed to tackle low power CNN inference [9]. Table 1 shows how different architectures exploit sparsity.

**Table 1: Comparison of Sparse CNN Accelerators**

| Architecture | Gate ALU op | Skip ALU op | Memory Access | Dataflow |
|---|---|---|---|---|
| Eyeriss[10] | A | - | A | Row Stationary |
| Cnvlutin[1] | A | A | A | Vector Scalar |
| Cambricon-X[11] | W | W | W | Dot Product |
| SCNN[8] | A+W | A+W | A+W | Cartesian Product |
| SparseCore | A+W | A+W | A+W | Vector Scalar |

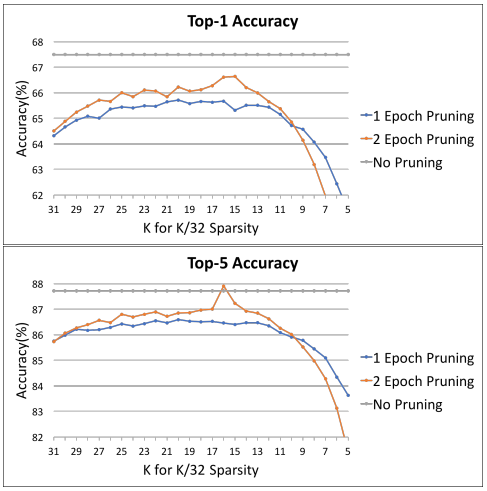



**Figure 1: Sparsity vs Accuracy trade-off for Resnet-18**

## 2 STRUCTURED SPARSITY

Early pruning techniques have been focused on reducing the total number of weights which should result in lower computational costs of inference. Such sparse layers end up having random sparsity structure among weight tensors resulting in inefficient design constraints for accelerators. In this paper, we propose a structured sparsity scheme. The basic motivation behind such a scheme is to enforce spatial structure on the layout of non-zero elements in compressed format.

### 2.1 *k/m:d* Sparsity

Given a $D$ dimensional tensor, the sparsity is defined along the vector in dimension $d$, as - *Out of every m elements, exactly k elements would be non-zero.* For convolution layers, $d$ is defined to be the dimension corresponding to the output depth.

### 2.2 Iterative Pruning

In order to observe the effect of sparsity on accuracy we experimented with ILSVRC2012 dataset. We iteratively prune the minimum weight in output dimension in CNN layers. We used ResNet-18

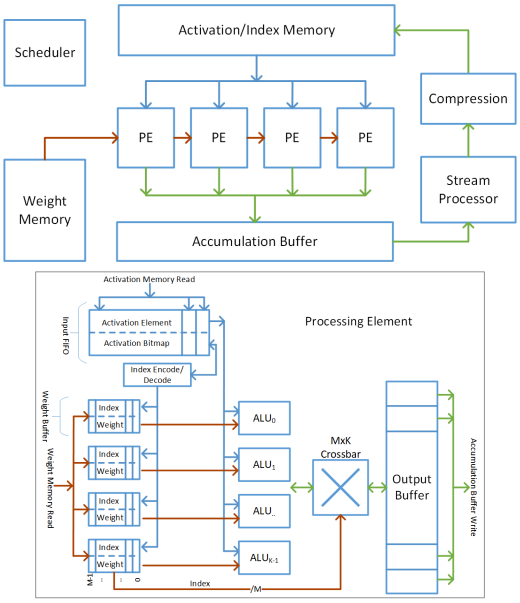



**Figure 2: SparseCore Architecture**

architecture for training. We first trained for 30 epochs thereafter, we iteratively pruned minimum weight in each layer to achieve the desired sparsity (Figure 1).

## 3 HARDWARE ARCHITECTURE

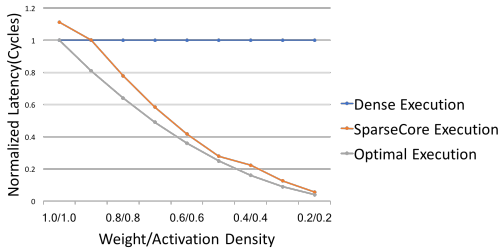
Majority of hardware accelerators utilize sparsity only in one tensor, either weights or activations. The choice of spatial and temporal unrolling of the data onto a particular accelerator also limits the ability to exploit sparsity fully. While ALU optimality is addressed in SCNN[8], the overhead of crossbars, memory barriers and work fragmentation limit the efficiency of SCNN especially in terms of energy utilization. SparseCore avoids such limitations by employing end-to-end global scheduling and dynamic assignment of work units.

### 3.1 Dataflow Architecture

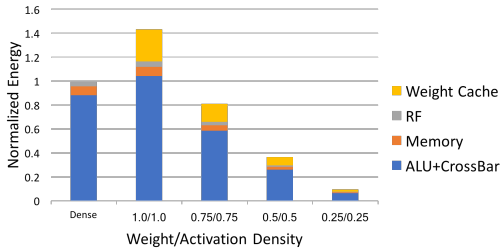
Since the network is compressed, the input bandwidth of the ALUs is likely to be overshadowed by uncompressed output bandwidth requirement. This leads us to propose a novel dataflow scheme called **Weight Local-Output Stationary**. Figure 2 shows microarchitecture of the SparseCore accelerator. The compute subsystem consists of array of Processing Elements (PE), where each PE is assigned an output depth chunk as a work unit. Intermediate partial summations are stored in the Accumulation Buffer, and upon completion, the data is sent through streaming processor for operations like pooling, normalization, RELU etc., and are eventually compressed and written to the output layer section of the Activation Memory. Table 2 gives detailed area breakdown for the accelerator with 64 PEs consisting of 16 ALUs each. On-chip memories contribute to about 65% of the area of the accelerator core.

**Table 2: SparseCore Area Breakdown**

| Accelerator Unit | Size | Area(sqmm) |
|---|---|---|
| PE Area | 64 | 2.43 |
| Activation/Index Memory | 2MB | 3.35 |
| Weight Memory | 512KB | 0.84 |
| Accumulation Buffer | 512KB | 0.84 |
| Misc | - | 0.2 |
| Total Area | | **7.66** |




**(a) Performance**




**(b) Energy**

**Figure 3: SparseCore Performance and Power**

## 4 RESULTS

Figure 3 gives performance and power details for a convolution layer with input volume of 64x64x64 and 3x3x128 filter. SparseCore can achieve near optimal performance in terms of ALU utilization while consuming significantly less energy compared to dense output stationary execution of the network.

## 5 CONCLUSION

This paper presents SparseCore architecture for CNN inference. SparseCore utilizes a novel $k/m$ structured sparsity scheme to implement optimal sparse computation. Iterative pruning methods can be used to reach high degree of $k/m$ sparsity while retaining the accuracy of the network.

SparseCore exploits both weight and activation sparsity using Weight Local-Output Stationary dataflow. This maximizes weight reuse and minimizes synchronization overhead between processing elements. In addition, both weights and activation can be stored in compressed format reducing on-chip storage requirements. Our analysis shows that with 50% weight and activation sparsity, SparseCore achieves performance improvement of 3.1x and reduces dynamic power consumption by 2.5x.

# REFERENCES


[1] Jorge Albericio, Patrick Judd, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (2016), 1–13.

[2] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.

[3] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael L. Seltzer, Geoffrey Zweig, Xiaodong He, Jason D. Williams, Yifan Gong, and Alex Acero. 2013. Recent advances in deep learning for speech research at Microsoft. In *ICASSP*. IEEE, 8604–8608.

[4] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015).

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

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. 1106–1114.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[8] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. *CoRR* abs/1708.04485 (2017). arXiv:1708.04485 http://arxiv.org/abs/1708.04485

[9] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[10] Yang, Tien-Ju and Chen, Yu-Hsin and Sze, Vivienne. 2017. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[11] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *MICRO*.