# STACKED FILTERS STATIONARY FLOW FOR HARDWARE-ORIENTED ACCELERATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS

**Gao Yuechao, Liu Nianhong & Zhang Sheng** *
Department of Microelectrics and Nanoelectrics
Tsinghua University
Beijing, 100084, China
{gyc15,lnh15}@mails.tsinghua.edu.cn

## ABSTRACT

To address memory and computation resource limitations for hardware-oriented acceleration of deep convolutional neural networks (CNNs), in this paper we present a computation flow, stacked filters stationary flow (SFS), and a corresponding data encoding format, relative indexed compressed sparse filter format (CSF), and also a three dimensional Single Instruction Multiple Data (3D-SIMD) processor architecture to take full advantage of these two features. Comparing with the state-of-the-art result (Han et al., 2016b), our method achieved 1.11× improvement in reducing the storage required by AlexNet, and 1.09× improvement in reducing the storage required by SqueezeNet, without loss of accuracy on the ImageNet dataset. Moreover, using this approach, chip area for logics handling irregular sparse data access can be saved.

## 1 INTRODUCTION

CNNs have achieved substantial progress during the past several years. But hardware resource limitations hinder their wide usage in embedded devices. Various efforts have been made to address this issue, such as ShiftCNN (Gudovskiy & Rigazio, 2017), Ristretto (Gysel, 2016), Eyeriss (Chen et al., 2017), Deep Compression (Han et al., 2016b) and EIE (Han et al., 2016a). Through compressing deep neural networks with pruning, trained quantization and Huffman coding, Deep Compression (Han et al., 2016b), the best paper of ICLR 2016, achieved state-of-the-art result in reducing storage requirement of neural networks without affecting their accuracy.

But manipulating compressed data consumes extra clock cycles and need considerable extra logics (about 19.1% chip area in (Han et al., 2016a) ). In this paper we present a novel computation flow SFS, and a corresponding memory layout and encoding format CSF, which make the best of data sparsity, and simplify data handling at execution time. We also propose a 3D-SIMD processor architecture which accelerates deep CNNs by taking advantage of SFS flow and CSF format.

## 2 STACKED FILTERS STATIONARY FLOW (SFS)

Computations in CONV layers and FC layers can be unified into a single formula (biases are ignored):

$$V_o[i][x][y] = \sum_{ch=0}^{C-1} \sum_{r=0}^{K-1} \sum_{c=0}^{K-1} W_f[i][ch][r][c] \times V_i[ch][Sx+r][Sy+c] \tag{1}$$

$V_o$, $V_i$ and $W_f$ are the matrices of the output feature maps, input feature maps and filters, respectively. $S$ is a given stride size. $C$ is the channel number, and $K$ is the filter kernel size.

Our approach (SFS) is to group the filters into batches with batch size $m$ ($M$ is filter number, $M' = M/m$), and reshape $W_f$ to $W_{f'}$. As showing in figure 1a. One channel of feature data will

*Corresponding author. zhang_sh@tsinghua.edu.cn

convolute with $m$ filter data from the same channel at a time. Each element in the feature map will multiply with a column of data from $m$ filters at the same position. The new computation formula is:

$$V_o[i \times m + j][x][y] = \sum_{i=0}^{M'} \sum_{ch=0}^{C-1} \sum_{r=0}^{K-1} \sum_{c=0}^{K-1} \sum_{j=0}^{m} W_{f'}[i][ch][r][c][j] \times V_i[ch][Sx + r][Sy + c] \quad (2)$$

| Filter 1 | W1,11 | W1,12 | W1,13 | W1,21 | ... | W1,33 |
|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... |
| Filter m | Wm,11 | Wm,12 | Wm,13 | Wm,21 | ... | Wm,33 |

a)

| Virtual weight value | W1,11 | W2,11 | ... | Wm,11 | ... | W1,kk | W2,kk | ... | Wm,kk |
|---|---|---|---|---|---|---|---|---|---|
| Relative filter index | 0 | 0 | ... | 0 | ... | 0 | 0 | ... | 0 |
| Relative column pointer | 0 | m | ... | m | m | | | | |

b)

Figure 1: Memory layout a) for $m$ filters with kernel size 3 from a single channel. b) in CSF format.

## 3 RELATIVE INDEXED COMPRESSED SPARSE FILTER (CSF) FORMAT

Several sparse matrix encoding formats have been proposed, such as CSC, CSR and CISR (Fowers et al., 2014). But existing encoding formats complex the computation at runtime due to its irregular memory access characteristics. This results in inefficiency in parallelizing computation and bigger chip area. Therefore, it would be desirable if the encoding can be directly used during run time. We introduce CSF encoding format to achieve this goal.

Our approach is to further rearrange the memory layout of the $m$ filters illustrated in figure 1a, storing the elements column by column, so after changing the computation flow using SFS approach, we could still load the filter weight sequentially at run time. The first line in figure 1b illustrates the changing. If there is any weight value equals to 0, just remove the column contains that value, and add one to the relative index in the next column, also subtract one to the pointer next to it. The non-zero value number (including padded zeros) of a column is given by the next pointer. Column pointer is 0 means all the values in the column before it equal to 0. Relative column pointer is not needed when parameters are stored in files.

## 4 3D-SIMD PROCESSOR ARCHITECTURE

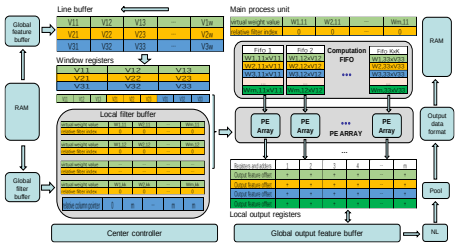


Figure 2: 3D-SIMD processor architecture.

Different hardware acceleration designs have been proposed, such as Eyeriss (Chen et al., 2017), DVAS (Moons & Verhelst, 2015), ENVISION (Moons et al., 2017), DNPU (Shin et al., 2017), etc. For deeply compressed sparse networks, the computational efficiency of these architectures is fairly low. We propose a 3D-SIMD processor architecture which takes full advantage of parallel computation and saves considerable clock cycles in handling data sparsity comparing to former architectures.

SFS flow and CSF encoding format are the key features of the proposed 3D-SIMD processor architecture, see figure 2. In this architecture, after feature data are loaded into the line buffer and window registers from a single channel of input feature map, and $m$ filter data from the same channel are loaded to the local buffer, each element in the window will multiply with a column of data from $m$ filters at the same position. So data in CSF format can be directly used at run time, and all zeros are skipped as designed. No complex sparse data handling logic is required.

Table 1: Storage requirement improvement

| Network | Nonzeros(bits) | Extra space (bits) | Improvement | Total |
|---|---|---|---|---|
| AlexNet by (Han et al., 2016b) | 30592056 | 41325956 | | |
| AlexNet by SFS+CSF | 30592056 | 34138970 | 1.21× | 1.11× |
| SqueezeNet by (Han et al., 2016b) | 3327368 | 1737628 | | |
| SqueezeNet by SFS+CSF | 3327368 | 1307160 | 1.33× | 1.09× |

## 5 RESULT

We first evaluated the distribution of continuous zero numbers after applying our method. As figure 3 shows, the distribution narrows to the left. That means it need fewer bits to store the relative index values, and there will be fewer padding zeros. This will further reduce storage space. We also evaluated the distribution of continuous nonzero numbers after applying the changes. The distribution also narrows to the left. That means the computation load balance during execution will be better comparing to former work. We also analyzed the effect of batch size $m$. We found that there do exist an optimum batch size for each layer. Smaller batch size requires smaller local buffer, but the input feature map need to be load more times. For simplicity, all the experiments in this section use filter number as the batch size. That means, all the filters and input feature maps are loaded only one time and the output feature maps are saved one time for one reference.

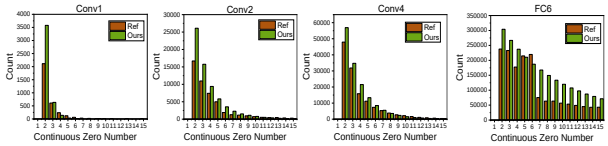


Figure 3: Distributions of continuous zero numbers, comparing with ref (Han et al., 2016b).

We use the following formula to find the best bit number to store the relative index values of each layer:

$$\arg\min_{bit} \{f_{total_bits}(bit) = Nz_num \times bit + \sum_{i=2^{bit}}^{\max} \text{zero}_stat[i] \times (\frac{i}{2^{bit}}) \times (wbit + bit)\} \quad (3)$$

$Nz_num$: the number of non-zero weight values; $wbit$: the number of bits to store weight value; $bit$: the number of bits to store relative index value; $zero_stat$: the distribution of continuous zero numbers.

Ttable 1 illustrate the improvement of storage requirement after applying our method comparing to former work.

## 6 CONCLUSION

In this paper, we propose a computation flow SFS, and its corresponding data encoding format CSF. And we also proposed a 3D-SIMD processor architecture to make the best of these two features. Experimental results show that our approach narrowed the distribution of the numbers of continuous zeros and non-zeros to the smaller count number direction. This helps to further compress the network parameters by about 8% to 10% and balance computation load at run time. By adopting the proposed 3D-SIMD processor architecture, chip area for logics handling irregular memory access of sparse data can be saved, for example, about 19.1% chip area in EIE (Han et al., 2016a) for pointer read can be saved. Directly using the encoded data as it is can also save zero bypassing clock cycles, and data locking up time( (Moons & Verhelst, 2015), (Moons et al., 2017)), (Han et al., 2016a)).

## REFERENCES

Yu Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

Jeremy Fowers, Kalin Ovtcharov, Karin Strauss, Eric S. Chung, and Greg Stitt. A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 36–43, 2014.

Denis A Gudovskiy and Luca Rigazio. Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks. 2017.

Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. 2016.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *International Conference on Computer Architecture (ISCA)*, 2016a.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016b.

Bert Moons and Marian Verhelst. Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. In *Ieee/acm International Symposium on Low Power Electronics and Design*, pp. 237–242, 2015.

Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *Solid-State Circuits Conference*, pp. 246–247, 2017.

Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi Jun Yoo. 14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *Solid-State Circuits Conference*, pp. 240–241, 2017.