Selective Data Transfer from DRAMs for CNNs

Anaam Ansari*†, Tokunbo Ogunfunmi*‡
*Department of Electrical Engineering
Santa Clara University, Santa Clara, California 95053

***Abstract*—Convolutional Neural Networks (CNN) have changed the direction of image and speech signal processing. They have becoming prolific in applications such as self driving cars and voice assitants like Siri and Alexa. Since the success of AlexNet, many deep learning networks like GoogleNet, ResidualNet etc have been introduced. These networks are highly competent in image classification however, they are very large in size and have a lot of parameters, for example, AlexNet has 60M parameters [1]. On-Off chip data transfer is a great engineering challenge that needs to be addressed while implementing CNN in hardware. Hardware Acceleration and parallelization are constrained by energy cost of reading and writing data from memory. For one forward pass during inference, one image needs to go through to all the CNN layers and be classified into a softmax determined category. During this process, the memory bandwidth is used by three types of payloads that need to be moved on and off-chip - input image, intermediate feature maps, and filter weights. This research is focused on reducing weight-related memory traffic that occurs between off-chip memory and on-chip buffers. In this paper, we propose a technique named 'weight sharing selective transfer' (WS-ST) that uses processing in-memory architecture to selectively transfer weights from the DRAM memory structure to the computation unit. We observe a 30% decrease in memory transfer traffic compared to a non-selective approach for AlexNet. We achieve this by implementing a selector logic in the DRAM itself. This logic helps us select the weights that need to be updated and avoid the transfer if the weight value is the same.**

***Index Terms*—CNN, selective data-transfer, AlexNet**

I. Introduction

Deep Learning is an emerging technology that is taking over new areas of application. Deep learning networks are evolving to be larger and more complex than ever [3], [4]. The hardware technology is constantly playing catch up to keep up with the sophistication we see in the algorithmic innovation of CNNs. Power and computation complexity are the challenges that need to be addressed as these networks keep getting larger and more complex. Hardware acceleration on FPGAs [5] and ASICs is gaining momentum as they prove to be more energy efficient and portable than GPUs for inference [6] [7].

Although hardware acceleration on FPGAs is lucrative, there are unique engineering challenges that limit this area. The major challenges include memory related traffic management, computation complexity, and complex parallelization. The memory bandwidth directly contributes to energy consumption as the movement of data from the DRAM is the most costly of all operations [2].

†aaansari@scu.edu
‡togunfunmi@scu.edu

TABLE I
Energy Cost of DRAM access compared to other operations [2]

Operation	Energy (pJ)	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit 32KB SRAM	5	50
32 bit DRAM	640	6400

There are many popular dense CNN Accelerator designs for FPGAs [8]–[10]. The techniques commonly used in dense architectures are loop blocking, loop unrolling and data stationary operations and data reuse.

In contrast to this, there has been new research in adapting sparse models and sparse computation in CNNs. Pruning techniques have proved that smaller models gain a lot in terms of performance and energy efficiency at a slight bearable cost in accuracy [2], [11]–[15].

In this work, we contribute the following:

- We analyze the redundancy in the filter weight kernels that are used in convolutional layers and fully connected layers after they are compressed using a technique called weight technique called 'weight sharing'.
- We model a DRAM that is capable of performing simple logic operations like compare or XOR.
- The modified DRAM helps us reduce the weight transfer from DRAM and main computational unit.

II. Motivation

Latest techniques like pruning and sparse computation have illustrated the benefits of reducing the model size. These compressed models eliminate the need to store full sized models with all the parameters and do not hurt the original accuracy [16]. Another way to minimize the storage of input and weights in the main memory is quantization. [17]–[19]. Although, quantization leads to a deterioration of accuracy a balance can be achieved between maintaining a low enough precision and tolerable accuracy.

Techniques to compress the network such as pruning lead to sparse networks and generate associative data as it is stored in other irregular formats. However, processing irregular networks is beyond the scope of this research.

There has also been an interest in the processing-in-memory (PIM) architecture and near-data-computation (NDC) to reduce the memory transfer from main memory to host processing unit [20]–[23]. Due to the popularity of stacked DRAMs

such as HMC and HBM [24], [25]. These DRAM architectures provide a promising throughput and latency performance. The through silicon vias (TSVs) have a higher bandwidth.

In this research, we exploit the redundancy of weights that is a result of a process called 'weight sharing' [26]. We design a weight selector logic to selectively send only the weights that are different than the ones used in the prior output f-map calculation.

Weight sharing can reduce the number of parameters drastically in a network [2]. They can be presented by using a small number of bits hence, there are a limited number of values the weights can assume. As a result of sharing, many values are repeated across different kernels. This is the redundancy we will exploit.

To do this, we aim to have a processing-in-memory architecture using stacked DRAM. The weight selector we design will bring in chunks of weight from the weight payload into registers near the data memory. Then the comparison of previous weight to the subsequent weight takes place and a decision is made. If the weights are found to be the same no action is taken and if they are different we update them in the buffer.

III. Analysis

In this section, we explain the rationale and analysis that lies at the foundation of our approach. The following analysis gives an insight about the superfluous repetitiveness of the shared weights. We perform this analysis on many levels of granularity i.e. bit precisions such as 1 bit, 4bit (a nibble) and 8bit (a byte). Studying the redundancy at various precision levels helps us select a bus size for the DRAM that will be best suited for our architecture and design. For this analysis, we use pre-trained AlexNet weights as our source for weight [27].

A. Weight Sharing

As a part of our analysis, we first convert the AlexNet weights into a shared format. Weight sharing is a pruning technique that is used to overcome a network being over-fitted. It was discovered mainly to combat over-fitting a network however, it helps compress a model too [11], [16], [26], [28]. A compressed model is retrained with the shared weights. Weight sharing makes retraining very efficient and also helps the models be more generalized and less over-fitted.

Making weights assume same values is of strategic value from an algorithmic point of view as well as the engineering point of view. We covert the weights to a 4 bit precision and 8 bit precision. This gives us 16 and 256 weight values respectively. Once the 16 and 256 weights are shared among all the layers (Convolutional and Fully connected layers) we implement the selection process.

B. Selective PassThrough

After the weights are shared, we can design a selection logic to pass through weights that need to be updated as opposed to writing all the weights to on chip memory without dealing with the redundancy. The logic to select weights from the weight payload is described in figure 1. We compare the subsequent weight with the previous weight and if it is found to be different we update it on chip if not we do nothing. We can choose to compare at any desired granularity.

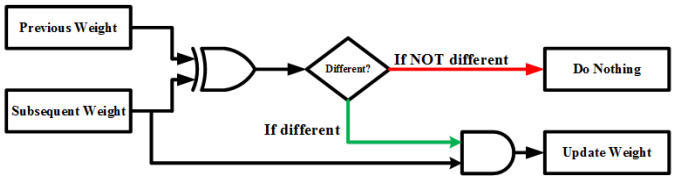


Fig. 1. Weight selector Logic

C. Redundancy Analysis

In the matlab analysis, we evaluate the redundancy in AlexNet weights after compressing them using weight sharing. The analysis process is as follows and entails the following experiments.

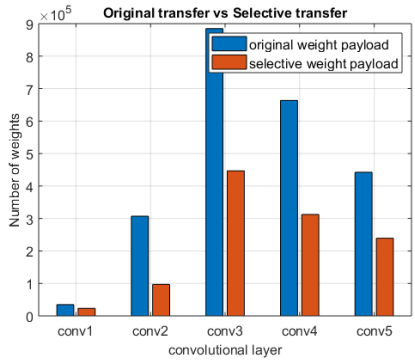
- Comparing 4 bit weights for 16 shared weights at the granularity level of 4 bits.
- Comparing 4 bit weights for 16 shared weights at the granularity level of 1 bit.
- Comparing 8 bit weights for 256 shared weights at the granularity level of 8 bits.
- Comparing 8 bit weights for 256 shared weights at the granularity level of 1 bit.

We choose 4bit and 8 bit weight precisions as 4bit of weight precision using dense sparse dense (DSD) training process in [15] does very well in terms of accuracy while also compressing it to a great degree. Looking at results in figures 2 and 3, we see that we are able to find more redundancy with smaller granularity. Both figures 2b and 2d in figure 2 show us that we can eliminate a large number of weight bits that are common between the batch of previous weight and the subsequent weight that needs to be sent to the chip. For fully connected layers, we see that in figures 3b and 3d we see the same trend. However, this is a very low level of granularity and will impose more challenges for a controller than being useful. One bit granularity will also provide a slow rate of transfer. Thus, the real choice is between 4 bit and 8 bit of granularity. Going to a higher granularity will increase the weight value levels and be in vain. Looking at figures 2a, 2c, 3a and 3c, we can see that 4 bit weights provide the right amount granularity and weight levels.

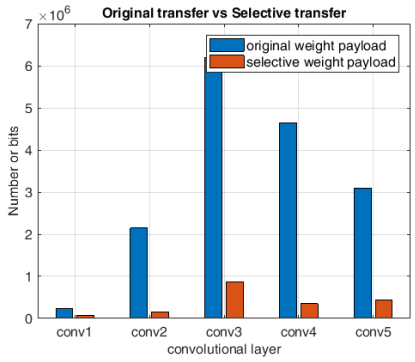
Looking at the result, we chosoe the 4 bit granularity as it exploits a good amount redundancy while being manageable in terms of hardware engineering. The savings with 4 bit weights and 4 bit granularity is described in table II.

IV. Design and Architecture

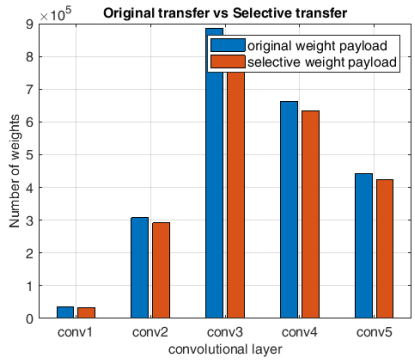
The contribution of this paper lies in the main memory. We reuse the systolic architecture with nested processing elements we developed in [29].



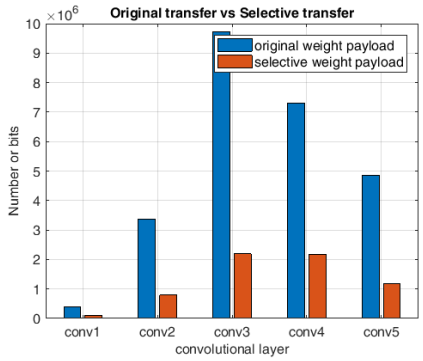
(a) 4bit weight, 4bit granularity



(b) 4bit weight, 1bit granularity

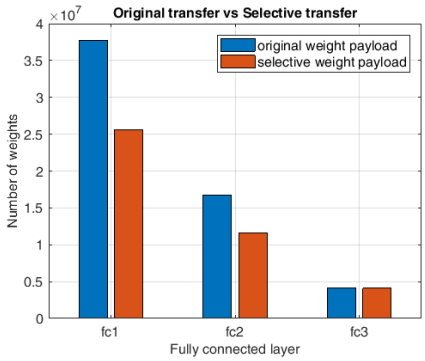


(c) 8bit weight, 8bit granularity

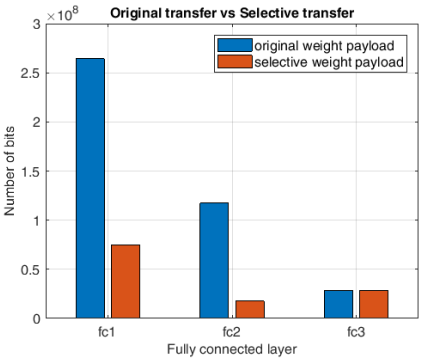


(d) 8bit weight, 1bit granularity

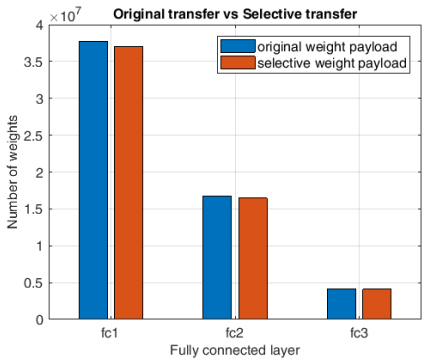
Fig. 2. Redundancy analysis of the Convolutional Layer Weights



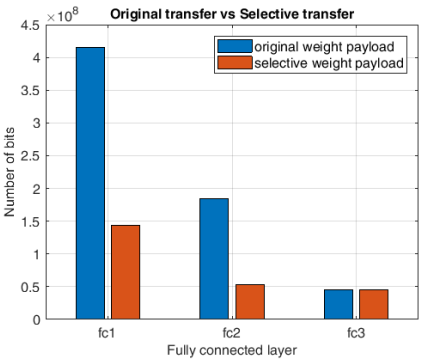
(a) 4bit weights, 4bit granularity



(b) 4bit weights, 1bit granularity



(c) 8bit weights, 8bit granularity



(d) 8bit weights, 1bit granularity

Fig. 3. Redundancy analysis of the Fully Connected Layer Weights

TABLE II
REDUNDANCY ANALYSIS FOR 4 BIT WEIGHTS WITH 4 BIT GRANULARITY

Layer	height	width	c	kernels	total weights	selective weights	difference	% Improvement
conv1	11	11	3	96	34848	23092	11756	33.74
conv2	5	5	48	256	307200	97045	210155	68.41
conv3	3	3	256	384	884736	446751	437985	49.50
conv4	3	3	192	384	663552	312345	351207	52.93
conv5	3	3	192	256	442368	239363	203005	45.89
fc1	4096	9216	1	1	37748736	25618179	12130557	32.14
fc2	4096	4096	1	1	16777216	11661838	5115378	30.49
fc3	1000	4096	1	1	4096000	4096000	0	0
total					60954656	42494613	18460043	30.28

A. Processing in Memory

Processing in memory or near data processing is done in a stacked DRAM. Stacked DRAM architectures are defined by [24], [25]. Each slice of DRAM is connected to each other with through silicon vias (TSVs). The DRAM is divided into smaller subsections called vaults. Each vertical vault is connected through and through, all the way to the logic layer. We implement the weight selector in the logic layer of the stacked DRAM.

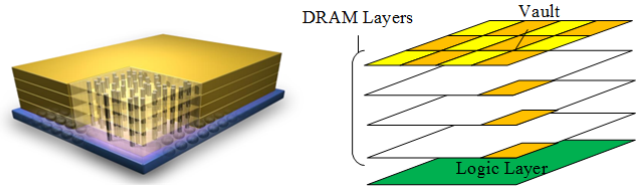


Fig. 4. HMC model

B. Weight Selector

We incorporate the processing in memory DRAM architecture into our systolic array architecture we designed for [29]. The main modification happens in the DRAM memory. The DRAM memory holds three kind of data payloads - input feature maps, filter weight kernel and output feature maps as described in figure 5. Along with the data payloads mentioned earlier we house a weight selector unit inside the memory so as to only transfer fresh and different weight updates and not repeatedly send values that might be in the buffers already.This is described in figure 6

We have to deal with the latency of sending first batch of weights to the computation. Once the computation layer is initialized with weights we need to perform weight selection so that we send only new updated weight.

V. POWER ESTIMATION

The authors in [30] have designed an online tool to estimate the energy consumption of DNN implementations using both computation and memory transfer cost. The values are normalized with respect to the energy cost of performing a MAC operation. We used the tool to estimate the energy consumption of the baseline AlexNet and the weight-sharing selective transfer AlexNet (WS-ST AlexNet). In figure 7, we see the energy consumption of the baseline AlexNet. In figure

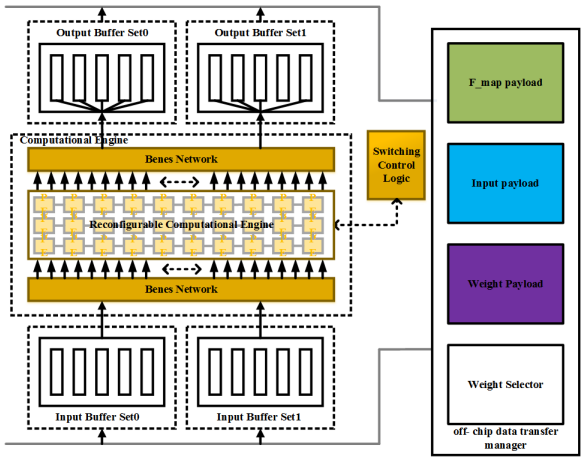


Fig. 5. Data Selector Architecture

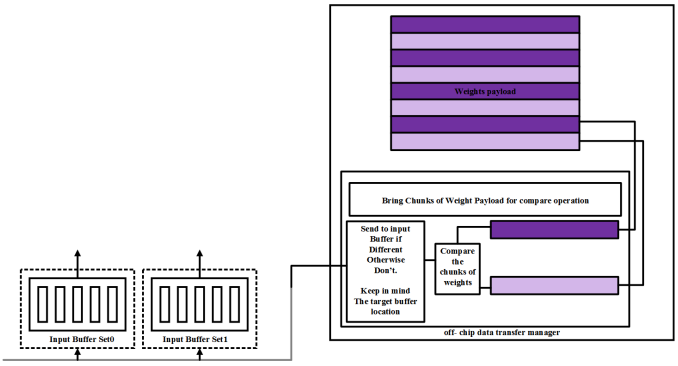


Fig. 6. Weight Selector Logic in DRAM

8, we see the estimated power consumption of the weight-sharing selective transfer AlexNet. This power estimation is for both weight related computation cost and memory transfer.

The energy consumption just due to weights is described in figure 9.

VI. HDL IMPLEMENTATION

We design a baseline DRAM and the WS-ST DRAM in Vivado and implement a design for the Virtex 7 family of devices. The results below in tables III and IV are from those implementations. We can see there is a saving in WS-ST dynamic and static power. If we probe further, we can see

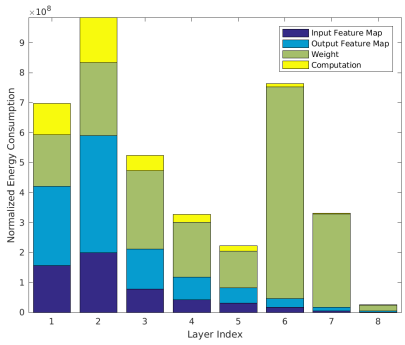


Fig. 7. Normalized Energy Consumption for AlexNet

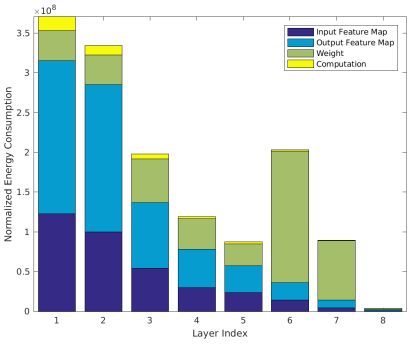


Fig. 8. Normalized Power Consumption for WS-ST AlexNet

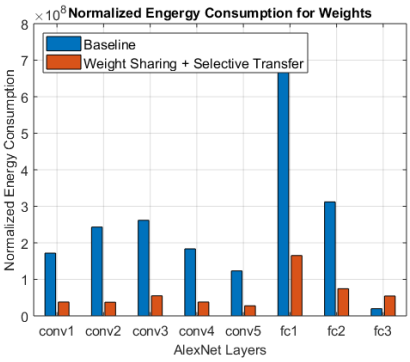


Fig. 9. Normalized Power Consumption due to weights

that the Logic power and IO power that constitute the dynamic power also see a reducing trend.

The memory width requirements for all layers are different. Fully connected layers have a large number of parameters. In our research, we find that storing fully connected layer parameters is difficult in a continuous block ram. Using a large continuous RAM increases the temperature of our target device to an unacceptable level during synthesis and implementation.

WS-ST architecture is not the best solution for fully connected layers. We will need to compress the fully connected layers weights using some other form of compression. This brings forward the need for a hybrid technique for convolutional layer processing and fully connected layer processing.

TABLE III
DYNAMIC POWER (DP) AND STATIC POWER (SP) FOR BASELINE ALEXNET AND WS-ST ALEXNET IN WATTS

	Baseline		Selective		% Improvements	
Layer	**DP (W)**	**SP (W)**	**DP (W)**	**SP (W)**	**DP**	**SP**
conv1	2.40	0.26	2.33	0.26	2.75	0.00
conv2	2.40	0.26	2.33	0.26	2.75	0.00
conv3	3.51	0.28	3.43	0.28	2.17	0.36
conv4	4.62	0.30	4.55	0.29	1.45	0.34
conv5	3.56	0.28	3.59	0.28	-0.87	-0.36

TABLE IV
LOGIC AND IO POWER FOR BASELINE ALEXNET AND WS-ST ALEXNET IN WATTS

	Baseline		Selective		% Improvements	
Layer	**IO (W)**	**Logic, (W)**	**IO (W)**	**Logic, (W)**	**IO**	**Logic**
conv1	1.788	0.016	1.684	0.031	5.82	0
conv2	1.788	0.016	1.684	0.031	5.82	0
conv3	1.844	0.164	1.749	0.163	5.15	0
conv4	1.844	0.178	1.749	0.193	5.15	0
conv5	1.69	0.104	1.602	0.128	5.21	0

VII. RESULTS AND CONCLUSION

We achieve a 30% reduction in the memory transfer of weights and 2% saving in dynamic power due to weights for AlexNet. Similar reduction could be observed in other popular networks. The fully connected layers really raise the energy consumption of the device thus we will require to formulate a new technique to store the fully connected layer parameters. In order to have a symmetry in architecture we will also need to implement a selection scheme on the output of the CNN layers. This will help us improve the offload data write and read to the main memory. Thus contributing to the overall of reduction in memory transfer traffic and consequentially a reduction in energy consumption.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] 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 *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 243–254.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

[5] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *arXiv preprint arXiv:1602.04283*, 2016.

[6] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145704

[7] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[8] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 247–257.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[10] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *IEEE Micro*, pp. 1–1, 2017.

[11] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[12] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, "SCNN: an accelerator for compressed-sparse convolutional neural networks," *CoRR*, vol. abs/1708.04485, 2017. [Online]. Available: http://arxiv.org/abs/1708.04485

[13] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," *arXiv preprint arXiv:1802.06367*, 2018.

[14] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv preprint arXiv:1705.08922*, 2017.

[15] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally, "Dsd: Regularizing deep neural networks with dense-sparse-dense training flow," *arXiv preprint arXiv:1607.04381*, 2016.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[17] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[18] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[19] S. Han, "Efficient methods and hardware for deep learning," University Lecture, 2017, May 25. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture15.pdf

[20] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 380–392.

[21] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 751–764. [Online]. Available: https://github.com/stanford-mast/nn_dataflow/

[22] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 2017, pp. 1–6.

[23] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[24] M. Black, "Hybrid memory cube," in *Electronic Design Process Symposium*, 2013, pp. 3–3.

[25] J. Standard, "High bandwidth memory (hbm) dram," *JESD235*, 2013.

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[27] Matlab. (2018) Pretrained alexnet convolutional neural network. [Online]. Available: https://www.mathworks.com/help/nnet/ref/alexnet.html

[28] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[29] A. Ansari, K. Gunnam, and T. Ogunfunmi, "An efficient reconfigurable hardware accelerator for convolutional neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, Oct 2017, pp. 1337–1341.

[30] T. J. Yang, Y. H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, Oct 2017, pp. 1916–1920.