A 200MHZ 202.4GFLOPS@10.8W VGG16 ACCELERATOR IN XILINX VX690T

Chunsheng Mei⋆ Zhenyu Liu† Yue Niu⋆ Xiangyang Ji‡ Wei Zhou⋆ Dongsheng Wang†

⋆School of Electronics and Information, Northwestern Polytechnical University, Xi'an, 710072, China.
†RIIT&TNList, ‡Department of Automation, Tsinghua University, Beijing, 100084, China
Email: liuzhenyu73,xyji,wds@tsinghua.edu.cn, zhouwei@nwpu.edu.cn

ABSTRACT

Convolutional Neural Networks (CNN) are among the most powerful and widely used algorithms for computer vision applications, notwithstanding their computation-demanding and memory-intensive operations. The cumbersome CNN operation stems from the bulky cross channel computation and massive parameter retrieving of convolutional (CONV) layers and fully-connected (FC) layers, respectively. In this paper, to remove the inter-filter redundancy, we constructed and tuned the specific low-rank filters in fully-connected layers. The proposed rank reduction saves 88.9% of both arithmetic and parameters of fully-connected layers in the VGG16 model. In addition, by employing network-layer-wise ping-pong DDR access mode, tile-grain on-chip feature map buffers, and *Propagate Partial Multiply-Accumulate* (*PPMAC*) processor, we implemented a 202.4 GFLOPS CNN accelerator with half-precision data format on Xilinx VC709 evaluation board. Experiments show that the accelerator achieved 6.58 fps throughput with 0.7046 top-1 accuracy and 0.8977 top-5 accuracy under 200MHz working frequency.

Index Terms— VGG16, FPGA, accelerator

1. INTRODUCTION

Convolutional Neural Networks (CNN), whose connection structure is inspired by the principle of mammalian visual cortex, belong to a category of feed-forward artificial network. The outstanding performance of CNN based algorithm is achieved by consuming a great many of computations and memory resources. For instance, VGG16 network, devised by Simonyan et al. [1], requires 138.34 million parameters when deploying it. One forward process of a 224×224 RGB image consumes 1.55 billion Multiply-Accumulate arithmetics. Because the general purpose processor is not efficient for 2-dimensional convolution operations, multiple CNN accelerators on the basis of GPU, ASIC, and FPGA have been developed [2–7]. As compare to GPU and ASIC counterparts, FPGA possesses the advantages of energy efficiency, reconfigurability and flexibility, which makes FPGA-based accelerator a promising approach in fast time-to-market and energy sensitive application scenarios. To maximize the CNN accelerator performance on FPGA, optimizations should be carried out from both aspects of algorithm and architecture design.

Extensive works [8–12] have been proposed to compress network parameters. A typical weight-pruning parameter compression algorithm, as in [8], contracted the ratio of non-zero parameters of VGG16 Model to 33.7% and 4.6% in convolutional layers and fully-connected layers, respectively. However, the irregular network parameter codec goes against pipelined computation of hardwired accelerator. Binary CNN network [12], which involves massive resource-comsuming 32 bit float point operations during scaling, is not a good choice for pipelined architecture either. Another kind of parameter compression scheme is to reduce the rank of parameter matrix [9, 10]. Though their compression ratio is lower than the previous weight-pruning algorithm, they are friendlier to pipelined computation. To improve the overall system performance, we exploited the redundancy of all FC layer parameters in VGG16 model.

Because the uneven computational overhead among different convolutional layers and and the data dependency between adjacent layers, the up-to-date CNN accelerator designs usually apply reconfigurable processing units with tiled input feature map to carry out all convolutions [3–5, 7, 13]. The primary difference of the architecture designs lies on the computation scheme, i.e., dataflow. Inspired by the motion estimation dataflow in video coding, we developed a PPMAC processor with half-precision data format to calculate 2-dimensional convolution efficiently. Based on this processor, an FPGA accelerator for VGG16 network is devised on Xilinx VC709 evaluation board, focusing on speeding up test-phase processing of VGG16 model. The accelerator is constructed in heterogeneous structure to speed up processing in picture level. Its performance is further improved by the reduced requirement of off-chip bandwidth resulted from aforementioned parameter compression.

The rest of the paper is organized as follows. Section 2 describes the VLSI friendly implementation optimization. The hardware architecture of our accelerator is presented in Section 3. The experimental results and comparisons with existing works are illustrated in Section 4, followed by the conclusions in Section 5.

2. VLSI FRIENDLY IMPLEMENTATION OPTIMIZATION

2.1. FC Layers Parameter Compression

Leveraging on the objective function to minimize reconstruction error of the ReLU activation function output, Zhang, et al. [10], derived the optimal low rank kernel matrix, targeting to remove the redundancy of all kernels in the same layer. Instead of remove redundancy in one layer, we exploit the redundancies of all the parameters in the FC layers.

Let $\{\mathbf{K}_i | i \in [14, 16]\}$ denote the set of all FC layer parameters at i-th layer of VGG16 network. Rank reduced matrix $\widetilde{\mathbf{K}}_i$ is generated by applying singular value decomposition (SVD) of $\mathbf{K}_i$, as shown in Equation 1.

$$\begin{aligned}\mathbf{K}_i &= \left(\mathbf{U}_i \cdot \mathbf{\Lambda}_i^{1/2}\right) \cdot \left(\mathbf{V}_i \cdot \mathbf{\Lambda}_i^{1/2}\right)^* \\ &\approx \mathbf{P}_i \cdot \mathbf{Q}_i = \widetilde{\mathbf{K}}_i.\end{aligned} \tag{1}$$

This work is funded by National Science and Technology Major Project (2016YFB0200505), Projects of International Cooperation and Exchanges NSFC (61325003), National Natural Science Foundation of China (61602383), Fundamental Research Funds for the Central Universities (3102016ZY024), Fundamental Natural Science Research Funds of Shaanxi Province (2016JM6077, 2017JQ6019) and Study Abroad Program of Northwestern Polytechnical University

Table 1. Energy concentration ς_i and channel number C

Layer#	Layer	ς_i	C
14	FC_4096	0.882	256
15	FC_4096	0.957	256
16	FC_1000	0.877	–

To determine parameter compression order of different layers, we kept 50% of the eigenvalues in $\mathbf{\Lambda}_i$ to get the approximated kernel matrix $\widetilde{\mathbf{K}}_i$. Then the corresponding energy concentration ς_i of these eigenvalues in each FC layer could be derived, as shown in Table 1.

According to descending order of ς_i, the channel number C of intermediate feature map is layer-wise determined. In specific, since the second FC_4096 layer (15th layer) possesses the highest degree of energy concentration, we compressed parameters of this layer first. Initially, by inserting 2048 1×1 intermediate channels, the associated kernel matrices $\mathbf{P}_{15}$, $\mathbf{Q}_{15}$ are 4096×2048 and 2048×4096 matrix, respectively. The initial value of $\mathbf{P}_{15}$ and $\mathbf{Q}_{15}$ are derived by SVD [14] of $\mathbf{K}_{15}$. Bias terms of $\mathbf{Q_{15}}$ are suppressed. The initial bias parameters of $\mathbf{P_{15}}$ are inherited from the original FC layer. Thereafter, we carried out the fine-tuning process of these two decomposed layer. The initial learning rate of layer $\mathbf{P}_{15}$ and $\mathbf{Q}_{15}$ are 10^{-4}, and decrease by a factor of 10 when the validation set accuracy stopped improving. Fine-tuning is stopped after 20K iterations. While the final prediction error difference ΔE is within the threshold 0.01 (original model is re-trained on ImageNet, with top-1 accuracy and top-5 accuracy be 0.6828 and 0.8837, respectively), we gradually removed the smallest eigenvalue in the reserved eigenvector until ΔE reaches 0.01 (when the compression ratio is not so large, this decomposing and fine-tuning scheme elevates network prediction accuracy). Next, under previously fine-tuned network model, we inserted 2048 intermediate channels in the 14th layer. And fine tuned these two layers by using the same strategy as in previous step. As the FC_1000 layer has lowest ς values, we kept the original parameter data as it is. To further utilize the computation resources in hardwired implementation, intermediate channel number C was enlarged to the nearest integral multiple of 32. The final results are listed in Table 1. We also fine-tuned on this modification before loading the network parameters to the accelerator.

2.2. Parallel Convolution Operation

The accelerator could speed up the convolutional processing by exploiting four kinds of parallelism, namely, Inter-Layer, Inter-Output-Feature-Map, Inter-kernel, and Intra-Kernel parallelism [15]. In the light of layer-wise convolution operation, Inter-Layer parallelism is impossible to realize. Meanwhile, the challenge of developing high performance FPGA accelerator comes from the limited on-chip memory and off-chip bandwidth of FPGA. When designing hardwired accelerator with high parallelism, these two factor shall be taken into account.

The pseudo code of our proposed CONV operation is shown in Fig 1. To realize multi-level parallel processing, we developed a PPMAC processor, which contains a group of Multiply-Accumulate (MAC) arithmetic unit. Due to the limited on-chip memory, we employed the tiled convolution. Convolution tile size is universally 14×14. To minimize data transportation, we rearranged kernel parameters so that the input tile could be disposed of after the complete computation of its contribution to each output channel. Since the granularity of tile is relatively small, the results could be accumulated in on-chip memories. The accelerator buffers up to 512 output feature map tiles. After the computation of the last input tile complete, the convolution results are dispatched to off-chip memories. In addition, rather than remaining to be idle waiting for current tile convolution finished, concurrently with the convolution operation, we prefetch next group of kernel parameter, or, tile from the next input channel provided that current kernel group is the last one. Data prefetching also ensures pipelined operation of the PPMAC processor.

```
function CONVOP(i, IBuf_x)
    C_o ← 0
    while C_o < O do
        if C_o = 0 and i = 0 then
            c ← 0
            READBIAS( )
            READKERNEL(C_o ~ C_o + 31, K_c)
        end if
        || PPMACCONV(K_c, IBuf_x)
        || PREFETCHKERNEL(C_o + 32 ~ C_o + 63, K_c)
        c ← c̄
    end while
end function

function CONV(T_x, T_y, C_i)
    T_x ← 0, T_y ← 0, C_i ← 0
    while T_y < Y do
        while T_x < X do
            while C_i < I do
                if T_x = 0 and T_y = 0 then
                    c ← 0
                    READTILE(IBuf_c, T_x, T_y, C_i)
                end if
                || CONVOP(C_i, IBuf_c)
                || PREFETCHTILE(IBuf_c, T_x, T_y, C_i + 1)
                c ← c̄
            end while
        end while
    end while
end function
```

Fig. 1. Pseudo codes of convolution layer operation (T_x, T_y denote current tile index in the input feature map; C_i/C_o is the input/output feature map channel index; $IBuf_c, K_c$ denotes one of the two input tile buffer and kernel parameter buffer (with buffer index c), respectively. $\|$ represents parallel processing)

3. ACCELERATOR ARCHITECTURE

3.1. System Architecture

The FPGA accelerator top block diagram is shown as in Fig 2. The initial statue, including the network model and the input images are transferred to the on-board DDR3 modules (DDR3 M0 and DDR3 M1) via PCIe3.0 x8. While forward processing, convolution layer and fully-connected layer have their own dedicated accelerators. An image grain pipeline is scheduled for FLAcc and CLAcc (When FLAcc is processing the fully-connected layers of current image, the convolutional computation of next image is being carried out by CLAcc). The proposed heterogeneous architecture is constructed on the condition that the convolution layers are suitable for parallelism, while the fully-connected layer are not. To further utilize on-board resources, two set of CLAcc, CLIB, CLOB, FLAcc and FLB were implemented to process two input image concurrently. Network parameters are shared between these two system accelerators.

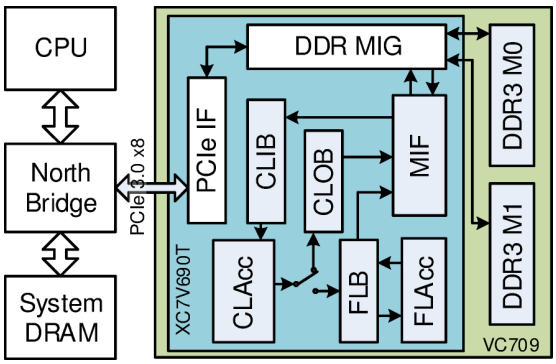

Fig. 2. Block diagram of system architecture (CLAcc: convolution layer accelerator; CLIB: convolution layer input buffer; CLOB: convolution layer output buffer; FLAcc: fully-connected layer accelerator; FLB: fully-connected layer buffer; MIF: memory interface; DDR MIG: off-chip memory interface provided by Xilinx; PCIe IF: PCIe interface provided by Xilinx)

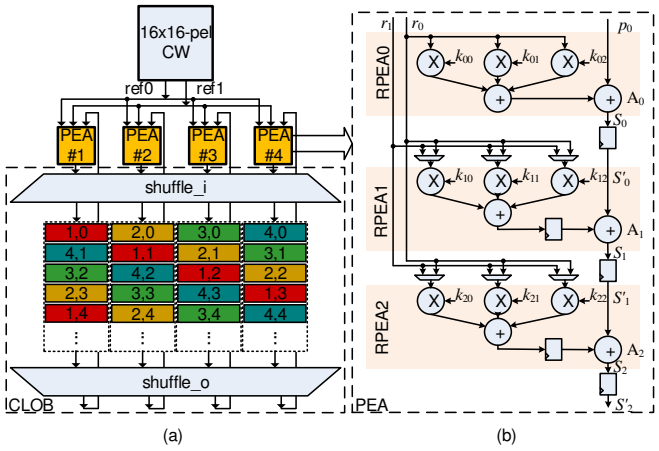

Fig. 3. CLAcc architecture ((a) Configuration for convolution operation with 4 PPMAC PEAs (4 PPMAC PEAs are used to demonstrate the procedure. Data items 'x, y', produced by PEA#x at clock y, are marked with different colors according to PEA's number); (b) PPMAC PEA.)

Each CLAcc is composed of 32 PPMAC architecture process element arrays (PEA) [16], 32 ReLU, pooling units. With 3×3 kernels and 14×14-pixel output feature map tiles, the input feature map patch is 16×16-pixel, which is denoted as convolution window (CW). The input pixel buffers, including one CW and the associated boundary pixel buffers, is denoted as CLIB. The component CLOB in Fig 2 is composed of 512 14×14-pixel buffers (implemented by using on-chip block RAM), where the number 512 is equal to the maximum output channel number. Fully-Connected Layer Buffer (FLB), containing $512 \times 7 \times 7$-pixels, is the inter-pipeline memory, to which the results of the last convolutional layer (13th layer in the VGG16 model) are dispatched. Because the computation loads of all fully-connected layers account for less than 1% of the overall computation loads, in our pipelined heterogeneous architecture, only one MAC unite is allocated for FLAcc.

3.2. CLAcc Architecture

For illustration purpose, Fig 3 (a) shows the concept of CLAcc Architecture with 4 PPMAC PEAs. The PPMAC PEA is provided in Fig 3 (b). In the design, two CLAccs are functionally duplicate of each other, simultaneously processing two tasks. We applied ping-pong storage scheme for input tiles and kernels, thus, the overhead of loading input tiles and kernels is ameliorated. Bias parameters are stored in CLOB. At the beginning of each CONV layer or FC layer operation, bias data are loaded as the initial value of accumulation. In addition, rather than using single-precision floating-point (32 bit) arithmetic, we adopted half-precision floating-point (16 bit) arithmetic for the accelerator as half-precision arithmetic provides enough accuracy for VGG16 model. The model parameters and input signals can be applied by our design without any network retraining. Besides, half-precision data format improves the deployment of our design, because it requires less off-chip bandwidth and on-chip resources comparing with single-precision data.

For each PPMAC PEA group, convolution input of CLAcc comes from one of the two CWs in CLIB. Suppose the output channel number of current layer is τ, it takes our CLAcc $\lceil \tau/32 \rceil$ rounds to process convolution. Because the critical path generated by the adder, The latency of one round is $14 \times 14 + 21 = 217$-cycle. In each round, all PEAs are devoted to their corresponding output channel's convolution. Let $I_i(R, C)$ denote the ith feature map's tile at (R, C) and r be the current round time, the jth PEA complete the task of

$$O_{32r+j}(R, C) = O_{32r+j}(R, C) + I_i(R, C) \otimes k_{i,32r+j}$$

where, $O_{32r+j}(R, C)$ is the $(32r + j)$th output feature map tile at (R, C), the symbol $\otimes$ denominates the convolution, and $k_{i,32r+j}$ is the kernel.

Inspired by the Motion Estimation hardwired engine [17], our 3-stage PPMAC PEA configuration is shown as in Fig 3(b) [16], which is composed of three row-wise 1D PE arrays (RPEAi, $i \in \{0, 1, 2\}$). p_0 is the partial sum read from CLOB. r_0 and r_1 are 1×3-pixel input vectors. r_0 fetches the top 14 rows in CW and r_1 is in charge of the last two rows. The data of r_1 are available only when the convolution position begins to change in column direction. The kernel coefficients are buffered in PEA. RPEAi computes the i-th row partial MAC value. It should be noted that the results of 3 RPEAi belong to 3 adjacent convolution positions. So, inter-stage registers are required to synchronize the partial MACs. The node S_2' outputs one 3×3 convolution result in each cycle after the initial 17-cycle latency. In each cycle, each PEA produces one pixel of output.

For the sake of routing and placement during FPGA implementation, the CLOB is composed of 32 3136×16 bit block memories instead of $512 \times 14 \times 14 \times 16$ bit distributed registers. During convolution, CLOB works as an intermediate accumulation buffer. We applied barrel shifting storage scheme for the buffer. Precisely, we right shift the 32 PEA outputs before it is loaded into the buffer. 32 outputs in the buffer RAM are shifted circularly by one position comparing to the outputs loaded at the previous clock. For instance, data produced by PEA#1 at clock 1, marked in red in Fig 3 (a), is shifted to the right with respect to data produced at clock 0. It is right shifted again at clock 2, and so forth. Because accumulation of these data are required, we need to shift back to counteract previous operations when retrieving them out. The *shuffle_o* is used to rearrange intermediate data read from the buffer. In the end, the output feature map tiles in the buffer are transferred to off-chip DRAM. Since the convolution results of one output channel's feature map are scattered in different block RAMs, with true dual port memories, one 7×7-pixel block could be extracted in one clock. Therefore, output data fetching latency is minimized.

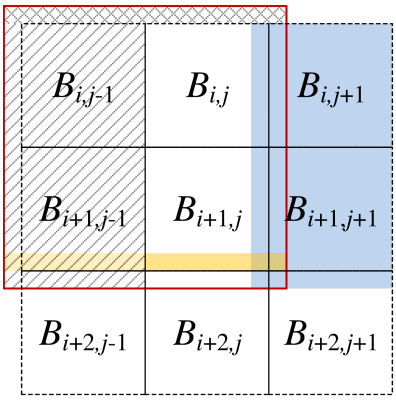

Fig. 4. Convolution Window caching strategy (Red block is 16×16-pixel CW; Each dot-line block represents one 7×7-pixel block)

3.3. CLIB Architecture

The CLIB module is composed of two 16×16-pixel CWs, one 1024×16-pixel top-row-buffer (TRB) and one $512 \times 15 \times 8$-pixel right-block-buffer (RBB). Two CWs realize the ping-pong access mode, so that the input tile reading can be merged in the computation. In addition to tiled input data, we applied 7×7-block-grain address mapping method. The pixels in one 7×7-block are stored in contiguous address. The off-chip DRAM module M0 stores the even convolution layers' input feature maps and the associated kernel filters; The odd convolution layers' input feature maps and kernel filters are deposited in M1. In this way, each DRAM module always maintains the same write- or read- mode, so that the transition latency between write- and read-modes is eliminated during processing.

To alleviate the burden of excessively off-chip memory accessing, we introduced TRB and RBB to cache the shared pixels of neighboring convolution tiles. The caching method of CW is illustrated as in Fig 4. Generally, while filling one CW, the 1×16 top row (the area filled with grid in Fig 4) is read from TRB. The 15×8 slash area is transferred from RBB. Then block $B_{i,j}$, $B_{i,j+1}$, $B_{i+1,j}$, $B_{i+1,j+1}$, and first 1×7-pixel rows of $B_{i+2,j}$ and $B_{i+2,j+1}$ are read from off-chip memory. When the CW is filled, the 1×16 pixels in yellow area replace the gridded area pixels in TRB, and the blue area substitutes the slash area in RBB. If CW is on input feature map boundary, the pixels in CW that outside the feature map are set with the default values. When CW is on the left boundary, $B_{i,0}$ and $B_{i+1,0}$ are read from off-chip memory. It should be noted that, since one 7×7 block is stored in two burst, the first row access of $B_{i+2,j}$ and $B_{i+2,j+1}$ only needs one burst operation. With the above schemes, the off-chip bandwidth utilization for input feature map reading is improved up to 65.6%.

4. EXPERIMENTS

Our accelerator is described with Verilog HDL. Xilinx Vivado 2016.2 Design Suite is adopted as the development EDA environment. The hardware platform is VC709 evaluation board, which is equipped with one Xilinx FPGA Virtex-7 XC7VX690T and two 4GB DDR3 memory modules. The clock speed of the CNN accelerator is 200 MHz, where the off-chip DDR3 memory modules are configured to work in 1600Mbps. The resource utilization of the overall design is exhibited in Table 2.

The system performance comparisons of our design and representative previous works are provided in Table 3. The theoretical

Table 2. FPGA XC7VX690T Resource Utilization

Resource	DSP	BRAM	LUT	FF	PCIe
Used	1728	196.5	210992	219538	1
Available	3600	1470	433200	866400	3
Utilization	48%	13.4%	48.7%	25.3%	33.3%

Table 3. Comparison with Other FPGA Accelerators

	[4]	[3]	Ours
Platform	XC7VX485T	XC7Z045	XC7VX690T
Clock(MHz)	100	150	200
Quantization Strategy	32-bit float	16-bit fixed	16-bit float
Power(W)	18.61	9.63	10.81
Processing Latency(ms)	–	163.42 (CONV) 224.60 (overall)	151.91 (CONV) 151.91 (overall)
Performance (GOP/s)	61.62	187.80 (CONV) 136.97 (overall)	202.03 (CONV) 202.42 (overall)
System Performance(fps)	–	4.45	6.58
Power Efficiency (W/fps)	–	2.16	1.64

latency T of one accelerator model could be written as

$$T = \sum_{i=1}^{13} \frac{W_i H_i c}{14 \times 14} \cdot 217 \cdot \Psi_i \cdot \frac{\Theta_i}{32}$$

where, Ψ_i and Θ_i represent the numbers of input and output feature maps of layer i, respectively; W_i and H_i are the width and height of the input feature map; c is one clock period. The processing latency is affected by T, the DDR data refreshing frequency, and the conflicts of external memory accessing. Experiments show that, the processing latency of CONV layers and FC layers are 151.91ms and 35.87ms per image processing, respectively. We can observed that the FC layer latency is 76.39% shorter than the CONV layer's counterpart. Because we applied picture level pipelined architecture, the maximum throughput is determined by the convolutional layer latency. With two set of system accelerators, the computational performance of our design is 202.42 GOP/s. Accordingly, the frame rate of our accelerator is 6.58fps, which is 1.47x times of the up-to-date results in literature [3]. Comparing to HLS design in previous works [3, 4], resource utilization and power consumption of our RTL oriented architecture are more efficient. With more instances of the accelerator, the system performance could be boosted up further more.

5. CONCLUSIONS

We developed a hardwired accelerator for VGG16 model on the platform of Xilinx VC709 evaluation board. With the parameter compression in fully-connected layer, 88.9% of both arithmetic and parameters are reduced. In the architecture design, we circumvented the hindrances of intensive computation in convolutional layers and huge off-chip bandwidth requirements in FC layers by using a picture grain heterogeneous pipelined structure. With off-chip bandwidth of 2.15GB/s, the proposed PPMAC convolution engine, which is composed of two $32 \times 3 \times 3$ process elements, achieved 202.03 GFLOPS at 200 MHz clock speed. On the other hand, a single MAC based FC layer engine has the competitive latency over convolutional engine. Leveraging on the above techniques, the proposed accelerator achieved 6.58fps throughput that outperforms previous FPGA based works.

6. REFERENCES

[1] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] NVIDIA Corporation, "NVIDIA cuDNN GPU accelerated deep learning," https://developer.nvidia.com/cudnn.

[3] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al., "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[4] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason 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.

[5] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen, "Cambricon-x: An accelerator for sparse neural networks," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[6] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh, "From high-level deep neural models to fpgas," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[7] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. ACM, 2015, pp. 92–104.

[8] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[10] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1984–1992.

[11] Shangzhen Luan, Baochang Zhang, Chen Chen, Xianbin Cao, Jungong Han, and Jianzhuang Liu, "Gabor convolutional networks," 2017.

[12] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," pp. 525–542, 2016.

[13] 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*, vol. 52, no. 1, pp. 127–138, 2017.

[14] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann Lecun, and Rob Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," pp. 1269–1277, 2014.

[15] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 575–580.

[16] Zhenyu Liu, Xianyu Yu, Yuan Gao, Shaolin Chen, Xiangyang Ji, and Dongsheng Wang, "Cu partition mode decision for hevc hardwired intra encoder using convolution neural network," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5088–5103, 2016.

[17] Ching Yeh Chen, Shao Yi Chien, Yu Wen Huang, Tung Chien Chen, Tu Chih Wang, and Liang Gee Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," vol. 53, no. 3, pp. 578–593, Mar. 2006.