A Novel Low-Communication Energy-Efficient Reconfigurable CNN Acceleration Architecture

Di Wu
State Key Laboratory of ASIC and System, Fudan University
15110720011@fudan.edu.cn

Jin Chen
State Key Laboratory of ASIC and System,Fudan University
15307130125@fudan.edu.cn

Wei Cao*
State Key Laboratory of ASIC and System,Fudan University
caow@fudan.edu.cn

Lingli Wang
State Key Laboratory of ASIC and System,Fudan University
llwang@fudan.edu.cn

***Abstract*—Winograd algorithm is an efficient approach to alleviate the computation burden of deep CNNs. Firstly, we introduce a fast matrix algorithm to combine with Winograd algorithm to further reduce the computation complexity and adapt the Winograd algorithm to large-stride convolution with a kernel-partitioning method. Secondly, computation efficiency improvement due to the fast algorithms aggravates the off-chip communication. DRAM access of different data-flows varies significantly with different CNN patterns. Dynamic configurations of both data-flows and on-chip shared memory can reduce the DRAM access effectively. A quantitative analysis is established on the design space to guide the configurations. Finally, a reconfigurable architecture that supports three categories of data-flows is presented. For evaluation, VGGNet16, AlexNet and ResNet50 are implemented respectively which can achieve the state-of-art DSP efficiency. Overall performance of 685.6GOP/s, 1250GOP/s and 507GOP/s for AlexNet, VGGNet16 and ResNet50 respectively on ZC706 platform and better energy efficiency are achieved compared with representative prior works.**

I. INTRODUCTION

CNNs are widely adopted in many artificial intelligence domains such as computer vision, speech recognition and video recognition. The state-of-the-art performance of CNNs is at the cost of immense computing workload as well as enormous memory burden. GPU is widespread and efficient for CNNs, but it is rarely applied in embedded systems owing to its high energy consumption. The FPGA-based approach has emerged as a promising solution due to its energy efficiency and flexibility.

Winograd algorithm has shown dramatic improvement on the CNN performance through the remarkable reduction of the computation complexity [1][2]. We introduce a fast matrix algorithm, which is orthogonal to the Winograd algorithm, to further reduce the computation complexity. Besides, we expand the application scope of Winograd algorithm to the large-stride convolutional layers with a novel kernel-partitioning method.

Fast algorithms improve the computation efficiency remarkably, which causes considerable tension for off-chip bandwidth capacity. Besides, off-chip communication requires several orders of magnitude higher energy than on-chip communication and computation [3]. Optimization of DRAM access is a necessity. Prior works tend to adopt a homogeneous data-flow [4] [5], which achieves small DRAM access in some layers but quite large DRAM access in other layers. Ref. [3] presents a survey for CNN accelerator data-flows but these effective data-flows are not combined. In our work, different data flows are utilized to satisfy diverse layer patterns. As to on-chip memory, separate buffers for input/kernel/output respectively [2][6] are replaced by on-chip shared memory. Since separate buffers require the maximal size among different layers, which degrades the memory utility and needs more DRAM access. Besides, a quantitative analysis is given to guide the configuration of data-flows and on-chip shared memory.

Fusion architecture [1] requires less feature-map transfer under the premise that kernel data of the layers in the fusion group should be stored on chip. However, such premise is not always satisfied especially for lightweight embedded systems.

In this paper, we make the following contributions.

(1) A fast matrix algorithm is combined with Winograd algorithm. A novel kernel-partitioning method is utilized to adapt the Winograd algorithm to large-stride convolutional layers (*Conv*).

(2) Three categories of data-flows and on-chip shared memory are applied to the diverse CNN patterns. A quantitative analysis guides the configuration.

(3) A reconfigurable architecture based on these data-flows is constructed. Both *Conv* and fully-connected *(FC)* layers can be implemented efficiently. VGGNet16, AlexNet and ResNet50 are validated as case studies.

II. BACKGROUND

A. Winograd Algorithm

$F(m \times m, r \times r)$ denotes computing $m \times m$ outputs with an $r \times r$ filter. Winograd documents the 2D minimal algorithm:

$$Y = A^T[(GfG^T) \odot (B^T iB)]A$$

where *f* is an $r \times r$ filter and *i* is an $(m + r - 1) \times (m + r - 1)$ input tile [7]. Multiplication complexity can be reduced from m^2r^2 to $(m + r - 1)^2$ with the Winograd method. *Wblock* is defined to describe the data region of $B^T iB$ (or GfG^T). The size of *Wblock*, *W*, equals $(m + r - 1)^2$.We target at filter 3×3 acceleration and choose $F(4 \times 4,3 \times 3)$ in our implementation.

B. A Fast Matrix Algorithm

A fast matrix algorithm can be utilized in the accumulation of results of different channels[8]. It achieves nearly $2\times$ acceleration by reducing computation complexity. D_1, D_2,…, D_N and R_1,…, R_M denote input data *Wblocks* and result *Wblocks*. *M* and *N* represent the number of output and input channels accordingly. R_j can be calculated as (2) and (3) in the conventional method and with the fast matrix algorithm respectively. The second term and third term of (3) can be shared. In addition, the third term and the fourth term can be computed offline to further reduce the on-chip computation.

$$R_j = \sum_{n=1}^{N} D_i \times K_{i,j} + bias_m, \qquad \mathrm{j} = 1,2,\dots,M \tag{2}$$

$$R_j = \sum_{n=1}^{\frac{N}{2}} (D_{2i-1} - K_{2i,j})(K_{2i-1,j} - D_{2i}) + \sum_{n=1}^{\frac{N}{2}} D_{2i-1}D_{2i} + \sum_{n=1}^{\frac{N}{2}} K_{2i,j}K_{2i-1,j} + bias_m, \quad j = 1,2,\dots,M \tag{3}$$

In order to calculate the first *Wblocks*, R_1 to R_M, of all the output feature maps *(outFMs)*, MN multiplications are needed in (2) while only $\frac{MN+N}{2}$ multiplications are needed in (3), almost half of (2) when both M and N are large. Suppose that *psum* and input/weight additions consume H and h hardware resources separately. Hardware resources used for additions in the fast algorithm vary from $(N-1)MH$ to $MNh + (\frac{MN+N}{2} + M)H$. Thus the fast matrix algorithm does not necessarily mean more hardware consumption for additions compared with the conventional method when the bit-width of *psum* is typically twice larger than that of input and kernel.

III. EXTENSION OF WINOGRAD ALGORITHM

A. Combination with the Fast Matrix Algorithm

Bit-width of input data is enlarged significantly after transformation matrices B while that of kernel data declines after transformation matrices G. $\sum_{n=1}^{\frac{N}{2}}(D_{2i-1} - K_{2i,j})(K_{2i-1,j} - D_{2i})$ and $\sum_{n=1}^{\frac{N}{2}} D_{2i-1}D_{2i}$ need many more bits than $\sum_{n=1}^{\frac{N}{2}} K_{2i,j}K_{2i-1,j}$. Directly combining the Winograd algorithm and the fast matrix algorithm causes heavy storage burden and extra computation expenses, so the transformation matrices B and G are revised to B' and G' respectively with a *ratio* for the storage convenience. Shift operations are used when *ratio* equals the nth power of 2 and *ratio* equals 8 in our implementation.

$$B' = \frac{B}{ratio}, G' = ratio \times G$$

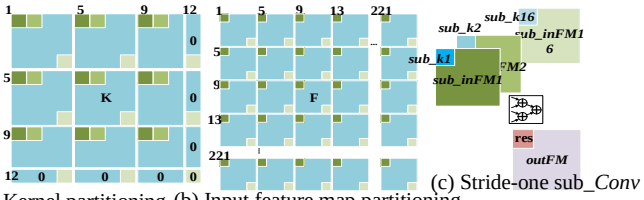
B. Winograd for large strides

Winograd transformation can be utilized by transforming large-stride *Conv* to stride-one *Conv* layers with a novel kernel-partitioning method. Assuming kernel window ($k \times k$), input feature map ($w \times w$) and stride length s, the initial large kernel is partitioned into X small sub-kernels (*sub_k*) and the large feature map is partitioned into X small sub-feature maps (*sub_inFM*).

$$X = s^2, k_s = ceil(\frac{k}{s}), w_s = \frac{w}{s}$$

where X is the number of sub-kernels and sub-feature maps, k_s is the size of sub-kernel, w_s is the size of sub-feature map.

As shown in Fig.1, *Conv1* of the AlexNet is taken as an instance, X equals 16. Fig.1(a) depicts the kernel partitioning. Firstly, zeros are padded on the right and bottom. The original kernel is expanded from 11×11 to 12×12. The expanded kernel is then divided to X sub-kernels. Each element from the same sub-kernel is fetched from the expanded kernel with the interval of stride length. Secondly, Fig.1(b) describes the similar way that input feature maps partition into X sub-feature maps. Thirdly, each partitioned kernel (*sub_ki*) and its corresponding partitioned feature map (*sub_inFMi*) constitute a stride-one convolutional sub-layer in Fig.1(c). Ultimately, convolution result from X sub-layers should be accumulated.



(a) Kernel partitioning (b) Input feature map partitioning (c) Stride-one sub_*Conv*

Fig. 1. Process of Transforming Large-Stride *Conv* to Stride-one *Conv*

IV. DESIGN SPACE OF DATAFLOW FOR DRAM ACCESS OPTIMIZATION

Three *CNN* computation categories (Winograd-based *Conv*, non-Winograd-based *Conv* with 1×1 filter and batch-based *FC*) can be abstracted into a uniformed representation, *Wblock* matrix multiplication in Fig.2. For 1×1 kernel *Conv* and batch-based *FC* layer, kernel data are copied W times and each kernel element in the same *Wblock* is the same. *Wblock* parallelism (P) denotes the number of *Wblocks* in each input feature map.

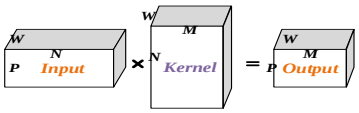
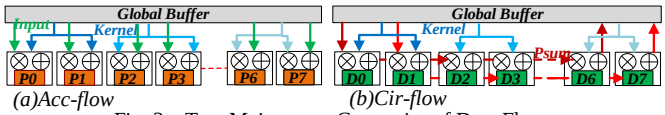


Fig. 2. *Wblock* Matrix Multiplication, *WMM*

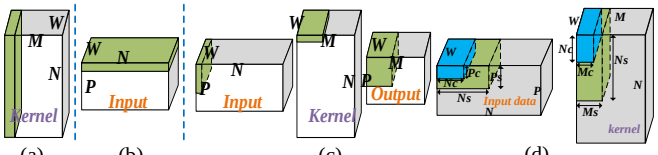


(a)Acc-flow *(b)Cir-flow*

Fig. 3. Two Mainstream Categories of Data Flows

Two data flows are proposed: *Acc-flow* and *Cir-flow*, which indicate accumulation and circulation of *psum* respectively. As shown in Fig.3(a), similar to Output Stationary (*OS*) [3], *Acc-flow* keeps the *psums* in the local register files (*RF*). *Acc-flow* differs from *OS* in that input data are conveyed in the multicast or the single-cast method rather than the stream form. In Fig.3(b), *Cir-flow* stores input data in the local *RF*. There're several PE groups to transport *psums* independently and kernel data are sent to the PE array in the multicast or the single-cast way.

In *Acc-flow*, on-chip global buffer denoted in green is mainly allocated to either input data or kernel in Fig.4(a) and Fig.4(b). Data tile of either input or kernel stored on chip cover all input channels, which helps make full reuse of input or kernel. Two variants of *Acc-flow*, named *Acc-flow1* and *Acc-flow2*, are explored. *Acc-flow1* stores all the kernel on chip when both the input channel and the output channel are relatively small. *Acc-flow2* stores a line of *Wblocks* of all input channels on chip when the input channel and output channel are slightly larger. More on-chip memory blocks are allocated to input data than kernel and input data are fully reused in *Acc-flow2* considering the permutation cost between the output data pattern and the required input data pattern of the next layer.



(a) (b) (c) (d)

Fig. 4. Storage Patterns: (a)*Acc-flow1*; (b)*Acc-flow2* and (c)*Cir-flow*. (d)Calculation Parallelism $<P_c, N_c, M_c>$

Input data of more *Wblocks* from the same feature map and kernel of more output channels can be stored on chip in *Cir-flow* in Fig.4(c). Large part of on-chip memory is to be allocated for *psum*. *Cir-flow* is adopted when the input channel is very large.

In Fig.4(d), <*Pc*,*Nc*,*Mc*> and <*Ps*,*Ns*,*Ms*> denote data size for computation unit and on-chip storage along the *P*,*N*,*M* dimensions respectively. *Wid* and *Dep* denote width and depth of on-chip shared memory. b_{in}, b_{ker} and b_{psum} represent bit-width of input, kernel and *psum* respectively. Configurations of both the computation unit and on-chip memory bandwidth are based on $<P_c, N_c, M_c>$. A quantitative analysis is presented to decide the optimal configurations <*Pc*,*Nc*,*Mc*> as follows.

TABLE I. DRAM ACCESS AND CONSTRAINTS OF THREE DATA FLOWS

	DRAM_Access	**constraints**
Acc-flow1 (*Ns=N*, *Ms=M*)	$W^2(\gamma_{in}PN + \gamma_{ker}NM + \gamma_{out}PM)$	$b_{in}N_cP_c + b_{ker}M_cN_c \le Wid$ $\frac{MN}{M_cN_c} \le Dep$
Acc-flow2 (*Ns=N*)	$W^2(\gamma_{in}PN + \gamma_{ker}NM\frac{P}{P_S} + \gamma_{out}PM)$	$b_{in}N_cP_c + b_{ker}M_cN_c \le Wid$ $\frac{NP_s}{N_cP_c} \le Dep$
Cir-flow (*Ns= Nc*)	$W^2(\gamma_{in}PN\frac{M}{M_S} + \gamma_{ker}NM\frac{P}{P_S} + \gamma_{out}PM)$	$2b_{psum}N_cP_c + b_{ker}M_cN_c \le Wid$ $\frac{M_sP_s}{M_cP_c} \le Dep$

With the fixed computation resources, $P_cM_cN_c = V_c$, where N_c is an even number because of the fast matrix algorithm. To minimize DRAM access, *Wid* and *Dep*, the theoretical and actual $<P_c, N_c, M_c>$ are listed in Table II where δ is the minimal nonnegative integer which makes $V_c \equiv 0 \pmod{N_c}$.

TABLE II. SUGGESTED $<P_c, N_c, M_c>$ OF THREE DATA-FLOWS (*VC* =32)

$<P_c, N_c, M_c>$	**Theoretical**	**Actual**
Acc-flow1	<1, 2, *Vc*/2>	<1,2,16>
Acc-flow2	< *Vc*/2, 2, *1*>	<8,2,2>
Cir-flow	$<\frac{V_c}{\lceil\frac{2b_{psum}}{b_{in}}\rceil+\delta}, \lceil\frac{2b_{psum}}{b_{in}}\rceil + \delta, 1>$	<4,8,1>

V. ARCHITECTURE DESIGN

A. On-chip Memory Hierarchy

All memory blocks are allocated for kernel storage in *Acc-flow1*. A quarter of on-chip memory blocks are allocated for kernel storage and half of the memory blocks are allocated for input data storage in *Acc-flow2* while *psums* are stored in *RF*. Assume the bit-width used for input data and kernel is 9 and the bit-width is 27 for *psum*, a quarter of global buffer (*GLB)* is allocated for kernel storage in *Cir-flow*, and the rest of *GLB* contributes to a double-buffer structure for *psum*.

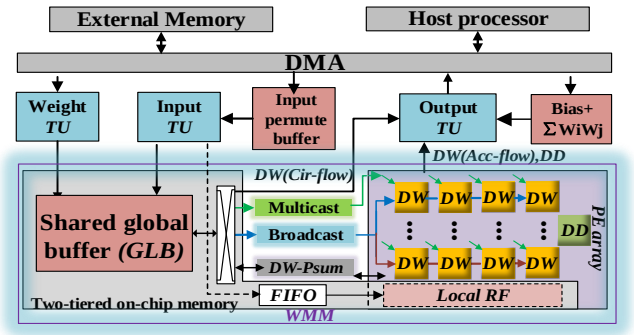
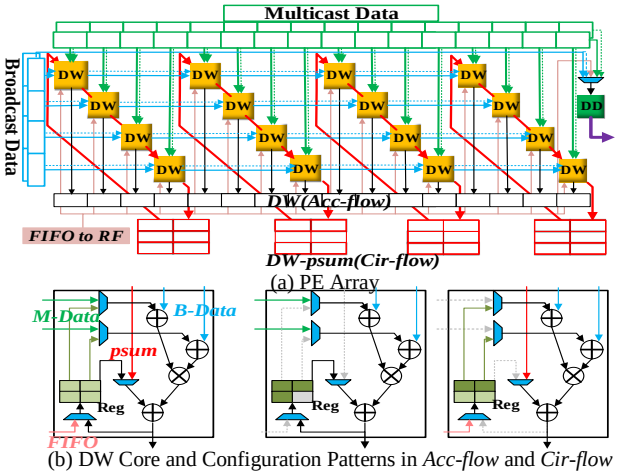


Fig. 5. Overall Architecture



(a) PE Array

(b) DW Core and Configuration Patterns in *Acc-flow* and *Cir-flow*

Fig. 6. Details of PE Array Design

B. Reconfigurable PE Array Design

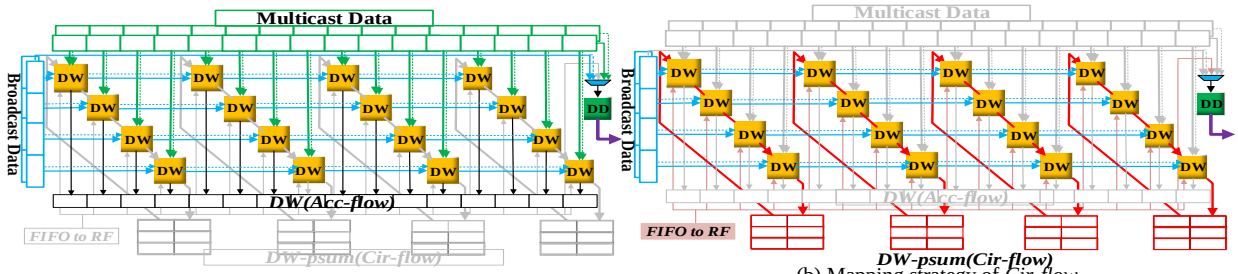
The basic structure of the PE array is presented in Fig. 6(a). It consists of 16 data-weight (DW) PE units and 1 data-data (DD) PE unit, computing the first two terms of (3) respectively. Four DW PE units in the same row share the same broadcast data. Multicast data consist of 16×2 channels and broadcast data consist of 4×2 channels. There are 36 DW cores in a DW PE unit and 36 multiply-accumulate units in a DD PE unit, in accordance to the size of *Wblock*. Three pieces of *RF* is used for *psum* storage in *Acc-flow*. Two pieces of *RF* store two input transformed data through FIFO, the other two constitute ping-pong structure in *Cir-flow*. (1) *Acc-flow mapping strategy:* All DW PE units work in parallel in Fig.7(a). Input data and kernel of sequential input channels are conveyed into the DW PE unit continuously and *psums* are accumulated. Multicast data are kernel data mapping 2 input feature maps (*FMs*) to 16 output *FMs*, and broadcast data are input data of 1 *Wblock* of 2 input *FMs* copied four times in *Acc-flow1*. In *Acc-flow2*, multicast data are input data of 8 *Wblock* of 2 input *FMs* copied twice and broadcast data are kernel data mapping from 2 input *FMs* to 2 output *FMs* copied twice. (2) *Cir-flow mapping strategy*: In Fig.7(b), input data are not transmitted as multicast data. Four *Wblocks* from 8 input *FMs* are stored in *RF*. Broadcast data are kernel data mapping 8 input *FMs* to 1 output *FM*. Four *psums* from the shared *GLB* stream along the DW PE units diagonally and then return to the *GLB* shown as red line. Different from *Acc-flow*, DW PE units in the different rows don't work concurrently. DW PE units in the second, third and fourth rows have one, two and three clock delays than the first row.

VI. EXPERIMENTAL EVALUATION

Our architecture is evaluated on Xilinx ZC706. The resource utilization after routing is shown in Table III. VGGNet, AlexNet and ResNet are implemented as case studies. Based on experiments with Ristretto[9], we find that 8-bit precision for feature maps and 8-bit (4-bit) precision for weights of *Conv* (*FC*) layers only introduce top1 and top5 accuracy loss of 1.5%,0.83% on average compared with the 32bit floating point.

TABLE III. RESOURCE UTILIZATION

Resource	**FF**	**LUT**	**DSP**	**BRAM(18k)**
This paper	94557	156859	612	667



(a)Mapping strategy of *Acc-flow* (b) Mapping strategy of *Cir-flow*

Fig. 7. Mapping Strategy of *Acc-flow* and *Cir-flow*

TABLE IV. PERFORMANCE COMPARISON OF VGG-16, ALEXNET AND RESNET-50

	VGG-16				AlexNet				ResNet-50		
	[5]	[1]	[2]	Ours	[6]	[2]		Ours	[11]	[10]	Ours
Device	ZC706	ZC706	ZCU102	ZC706	GSD8	ZC706	ZCU102	ZC706	GSMD5	GX1150	ZC706
Frequency(MHz)	150	100	200	150	120	167	200	150	150	200	150
CNN avg. (GOP/s)	137	-	2940.7	1249.7	72.4	201.4	1006.4	685.6	226.47	619.13	507.2
DSP Efficiency(GOP/s/DSPs)	0.152	0.28	1.16	2.04	0.037	0.224	0.339	1.12	0.22	0.408	0.829
Power(W)	9.6	9.4	23.6	9.82	19.1	9.4	23.6	9.76	25	-	9.72
Energy Efficiency (GOP/s/W)	14.3	24.42	124.6	127	3.79	21.4	36.2	70.2	9.06	-	52.2

The Singular Value Decomposition (SVD) is applied to the batch-based *FC* implementation. Batch-based *FC* can be implemented as two sub-layers where *Cir-flow* is adopted for the first sub-layer and *Acc-flow* for the second sub-layer. When the singular value number is set 512, 0.31% accuracy loss is introduced. We can achieve 23× speedup for *FC* layers.

DRAM access of each layer in VGG-16 acquired by three data-flows respectively are shown in Fig.8. The data-flow choices of each convolutional layer are highlighted with red points. We improve the DSP efficiency from 1.16GOP/s/DSP to 2.04 GOP/s/DSP in Table IV. We obtain even better energy efficiency than UltraScale-series ZCU102 implementation [2].The first layer of Alexnet has large stride and large kernel. With kernel partitioning method, Winograd algorithm is applied to the first layer and its computation load is reduced significantly. We improve the DSP and energy efficiency from 0.3GOP/s/DSP to 1.1GOP/s/ DSP, 36.2GOP/s/W to 70.2 GOP/s/W respectively in Table IV. With Winograd algorithm for 3×3 filter and the fast matrix algorithm for both 1×1 and 3×3 filters, we improve the DSP and energy efficiency of ResNet from 0.4GOP/s/DSP to 0.8GOP/s/DSP, 9.06GOP/s/W to 52.2 GOP/s/W respectively.

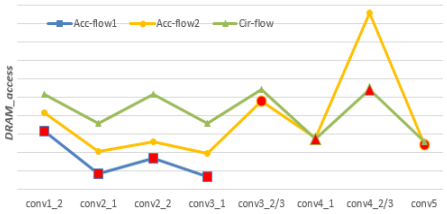


Fig. 8. DRAM_access of Each Layer in VGG-16 Implemented with Different Data-flows: *Acc-flow1*; *Acc-flow2* and *Cir-flow*

VII. CONCLUSIONS

In this paper, a fast matrix algorithm is introduced to combine with Winograd algorithm. We also adapt the Winograd algorithm to the large-stride convolution. To minimize the DRAM access, three categories of data-flows are designed for different CNN patterns. With a quantitative analysis, the optimal configurations are explored. A reconfigurable architecture that supports diverse data-flows is proposed. Our implementation achieves the state-of-art DSP efficiency, overall performance of 685.6GOP/s, 1250GOP/s and 507 GOP/s for AlexNet, VGGNet-16 and ResNet-50 respectively on ZC706.

REFERENCES

[1] Q. Xiao and et al, “Exploring heterogeneous algorithm for accelerating deep convolutional neural networks on FPGAs”, in DAC,2017

[2] L. Lu, Y. Liang, Q. Xiao, S. Yan, “Evaluating fast algorithms for convolutional neural networks on FPGA”, in FCCM, 2017

[3] V. Sze, Y.H. Chen, T.J. Yang, J. Emer, “Efficient processing of deep neural networks: a tutorial and survey”, arxiv: 1703.09039v1, 2017

[4] C. Zhang and et al. ”Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks”, in ICCAD, 2016

[5] J.Qiu and et al, “Going deeper with embedded FPGA platform for convolutional neural network”, in FPGA, 2016

[6] N. Suda and et al, “Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks”, in FPGA,2016

[7] S. Winograd. “Arithmetic complexity of computations”, Volume 33. Siam, 1980

[8] Richard E Blahut. Fast algorithms for digital signal processing. Addison-Wesley Pub.Co, 1985

[9] P.M.Gysel, “Ristretto: hardware-oriented approximation of convolutional neural networks”, arxiv: 1605.06402v1, 2016

[10] Yufei Ma, and et al. “An Automatic RTL Compiler for High-Throughput FPGA Implementation of Diverse Deep Convolutional Neural Networks”, in FPL, 2017

[11] Y. Guan and et al.“FP-DNN: An automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates”, in FCCM, 2017