Parallel and Reconfigurable VLSI Computing (13)

Deep Neural Network on an FPGA

Hiroki Nakahara Tokyo Institute of Technology

10.1 Al Trends

Artificial Intelligence is everywhere





Example: K-Glass



https://www.youtube.com/watch?v=fzQpSORKYr8

Intelligence and Deep Learning

Brain Inspired



J. Park, "Deep Neural Network SoC: Bringing deep learning to mobile devices," Deep Neural Network SoC Workshop, 2016.

Brief History: DNNs



Accuracy of a DNN



O. Russakovsky et al. "ImageNet Top 5 Classification Error (%)," IJCV 2015.

Why Deep Neural Networks?





Big Data

Computational Power

Computational Power and Big Data



High performance computation, big data, and a progress of Algorithms

(Left): "Single-Threaded Integer Performance," 2016

(Right): Nakahara, "インターネットにおける検索エンジンの技術動向(In Japanese)," 2014

DNN Frameworks



Academic…CAFFE, Industry…Tensorflow

Source by nVidia Corp.

CNN Applications

- Image:
 - Classification, Object detection, action recognition, scene understanding
- Natural Language:
 - Speech recognition, Translation
- Video:
 - Pedestrian detection, traffic sign recognition
- Medical:
 - Breast cancer cell detection, brain image segmentation







Application: Object Detection



Deep Learning for Embedded System

• Robot, Self-driving, Security Camera, Drone











Requirements

Cloud	Edge
Many classes (1000s)	Few classes (<10)
Large workloads	Frame rates (15-30 FPS)
High efficiency (Performance/W)	Low cost & low power (1W-5W)
Server form factor	Custom form factor

J. Freeman (Intel), "FPGA Acceleration in the era of high level design", 2017

10.2 High performance deep neural network

Computation Requirements for DNNs



• 20 Billion MACs (Multiply ACcumulation operation)/image

J. Park, "Deep Neural Network SoC: Bringing deep learning to mobile devices," Deep Neural Network SoC Workshop, 2016. J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *Artificial Neural Networks and Machine Learning (ICANN2014)*, 2014, pp. 281-290.

Artificial Neuron (AN)



y: Output signal

Deep Neural Network



LeNet-5

- Baseline 5 layer CNN by LeCun
- Convolutional (Feature extraction)
- Fully connection (Classfication)



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998. 19









• 2D computation of the AN operation



AlexNet

- ILSVRC'12 Winner (Error rate 16%)
- Augmentation
- 8 layer, dropout, ensemble CNN, rectified linear unit (ReLU)



A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012. 25

AlexNet vs LeNet-5

LeNet-5, 1998





#training images: 10⁷

NIST

AlexNet, 2012





#training images: 10¹⁴

IM GENET

26

Augmentation

- Increases # of training data
- Trade off: Memory and error rate



Flip horizontally



Color jittering



Random crops/scales

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)

GoogLeNet

- Network-in-network
- ILSVRC'14 winner (Error rate 6.7%)
- 22 layer, Inception, no-fully connection



VGG-16

• ILSVRC14 2nd place

(Error rate 7.4% \rightarrow 0.6 point worse)

- All conv. layer has the same kernel size (K=3)
- Variations: VGG11, VGG19



K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556

29

VGG Strategy: Going Deeper

- Upper layer ightarrow Distinct data
- Deeper CNN improves error rate



Visual Recognition", In Proc. ICML, 2014.

5

The deeper network...

- It has higher training error
- Backward: $(0.1)^{100} \rightarrow 0$, forward: $(1.1)^{100} \rightarrow \infty$



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 31

ResNet

- ILSVRC'15 winner (Error rate 3.57%)
- 152 layer, Batch normalization



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 32

Hardware Realization for CNN



Hardware Friendly CNN

	AlexNet	VGG	GoogLeNet	ResNet
Year	2012	2014	2014	2015
#Layers	8	19	22	152
Accuracy	16.4%	7.3%	6.7%	3.57%
Inception (NinN)			~	
Kernel Size	11,5,3	3	7,1,3,5	7,1,3,5
FC Size	4096,4096, 1000	4096,4096, 1000	1000	1000
Normalization	Local Response		Local Response	Batch

3.5% error improvement vs complex (different) and large HW $\rightarrow \textbf{VGG}$ is suitable

Binarized Neural Network

- 2-valued (-1/+1) multiplication
- Realized by an XNOR gate

x1	x2	Y
-1	-1	1
-1	+1	-1
+1	-1	-1
+1	+1	1

x1	x2	Y
0	0	1
0	1	0
1	0	0
1	1	1

Binarized CNN by XNORs



Binarized CNN by XNORs



Memory Size Reduction by Binarization (VGG-11)



38

Higher Power Efficiency

• Distance for the memory and ALU \propto Power

 \rightarrow On-chip memory realization



E. Joel et al., "Tutorial on Hardware Architectures for Deep Neural Networks," MICRO-49, 2016. 39

On-chip Memory Realization

- FPGA on-chip memories
 - BRAM (Block RAM) \rightarrow 100s \sim 1,000s
 - Distributed RAM (LUT) ightarrow 10,000s \sim 100,000s
- \rightarrow Small size, however, wide band

Cf. Jetson TX1(GPU) LPDDR4, 25.6GB/s

10,000@100MHz \rightarrow 125GB/s





Error Rate Reduction

• Introduce a batch normalization



H. Nakahara et al., "A memory-based binarized convolutional deep neural network," FPT2016, pp285-288, 2016.

High-level Synthesis Design for the Binarized CNN

Design Gap

Y = X.dot(W) + B



Python programmer writes only a single sentence module hoge(input a, b, c; output d); assign d = (a & b) | c; endmodule



HDL designer writes many gates

#Lines∝Design Time

Y = X.dot(W) + B

Python: single line

```
for(i=0;i<2;++i){
    for(j=0;j<2;++j){
        y[i][j] = x[i][j] * w[i][j];
        // Compute terms
        for(i=0;i<2;i++){
            for(j=0;j<2;j++){
                term = 0;
                for(k=0;k<2;k++)
                    term = term + x[i][k]*w[k][j];
                y[i][j] = term;
        }
    }
}</pre>
```

C/C++: 10 lines

input clk, reset, input [7:0]x[0:3], output [7:0]y[0:3] reg [1:0]state; reg [1:0]mux1, mux2; reg [7:0]w0, w1; reg [1:0]de_mux; always@(posedge clk or posedge rst)begin if(rst == 1'b1)begin state <= 2'b00 end else begin case(state) 2'b00:begin state <= 2'b01;</pre> mux1 <= 2'b10;</pre> mux2 <= 2'b11;</pre> w0 <= 8'b00101000; w1 <= 8'b11000101; de_mux <= 2'b11;</pre> end 2'b01:begin state <= 2'b10;</pre> mux1 <= 2'b00;</pre> mux2 <= 2'b01;</pre> w0 <= 8'b00101000; w1 <= 8'b11000101; de_mux <= 2'b00; end 2'b10:begin state <= 2'b11;</pre> mux1 <= 2'b00;</pre> mux2 <= 2'b01;</pre> w0 <= 8'b00101000; w1 <= 8'b11000101; de_mux <= 2'b01;</pre> end 2'b11:begin state <= 2'b00;</pre> mux1 <= 2'b00;</pre> mux2 <= 2'b01;</pre> w0 <= 8'b00101000; w1 <= 8'b11000101; de_mux <= 2'b10;</pre> end endcase end end wire [15:0]mul1, mul2; wire [16:0]w_add; assign mul1 = w0 * mux(mux1,x[0],x[1],x[2],x[3]); assign mul2 = w1 * mux(mux2,x[0],x[1],x[2],x[3]); assign w_add = mul1 + mul2; assign y[0] = (de_mux == 2'b00) ? w_add : 2'bzz; assign y[1] = (de_mux == 2'b01) ? w_add : 2'bzz; assign v[2] = (de mux == 2'b10) ? w add : 2'bzz;assign y[3] = (de_mux == 2'b11) ? w_add : 2'bzz; endmodule

module mat_add(

Verilog-HDL: 66 lines 44

High-Level Synthesis (HLS)

- C/C++ based for the software programmer
- 1 week for the NES
- 1 month for the binarized CNN





System on Chip FPGA



Source: Xilinx Inc. Zynq-7000 All Programmable SoC

Conventional Design Flow for the SoC FPGA



- 1. Behavior design
- 2. Profile analysis
- 3. IP core generation by HLS
- 4. Bitstream generation by

FPGA CAD tool

5. Middle ware generation

System Design Tool for the SoC FPGA



- 1. Behavior design
 - + pragmas
- 2. Profile analysis
- 3. IP core generation by HLS
- 4. Bitstream generation by FPGA CAD tool
- 5. Middle ware generation
 ↓
 Automatically done

A GUI-based binaryzed Neural Network Synthesizer (GUINNESS) Tool Flow

See, https://github.com/HirokiNakahara/GUINNESS



10.4 Implementation

Comparison with Other FPGA Realizations

Implementation (Year)	Zhao et al. [1] (2017)	FINN [2] (2017)	Ours
FPGA Board	Zedboard	PYNQ board	Zedboard
(FPGA)	(XC7Z020)	(XC7Z020)	(XC7Z020)
Clock (MHz)	143	166	143
#LUTs	46900	42833	14509
#18Kb BRAMs	94	270	32
#DSP Blocks	3	32	1
Test Error	12.27%	19.90%	18.20%
Time [msec]	5.94	2.24	2.37
(FPS)	(168)	(445)	(420)
Power [W]	4.7	2.5	2.3
FPS/Watt	35.7	178.0	182.6
FPS/LUT	35.8x10-4	103.9x10-4	289.4x10-4
FPS/BRAM	1.8	1.6	13.1

Y. Umuroglu, et al., "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *ISFPGA*, 2017. R. Zhao et al., "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," *ISFPGA*, 2017.

Comparison with Embedded Platforms (VGG11 Forwarding)

Platform	CPU	GPU	FPGA
Device		Jetson TX1 Developer Kit Jetson TX1 Developer Board 5MP Camera	
	ARM Cortex-A57	Maxwell GPU	Zynq7020
Clock Freq.	1.9 GHz	998 MHz	143.78 MHz
Memory	16 GB eMMC Flash	4 GB LPDDR4	4.9 Mb BRAM
Time [msec] (FPS)	4210.0 (0.23)	27.23 (36.7)	2.37 (421.9)
Power [W]	7	17	2.3
Efficiency	0.032	2.2	182.6
Design Time [Hours]	72	72	75

Conclusion

- DNN design methods for an FPGA
 - Binarized DNN \rightarrow Small hardware with high performance
- With high-level synthesis design
 - Short TAT
 - Considerable HW consumption
- Comparison with other FPGA based design
- Future works
 - Acceleration for the learning
 - Detection hyper parameters \rightarrow Grid search?

Exercise

 (Mandatory) Do HLS of the LeNet5.cpp on github: https://github.com/HirokiNakahara/FPGA_lecture/blob/master/ Lec13_DeepLearning/LeNet5.cpp

Then, report HW resources and its performance (clock freq. and latency)

My e-mail address: nakahara@ict.e.titech.ac.jp

Deadline is 31st, Aug., 2018 (include the final report)

Final Report

- 1. (Mandatory) Accelerate the LeNet5.cpp by inserting appropriate #pragma
 - 1. (Optional) Run a C++ simulation by modifying the source code (Note that, Vivado HLS report a simulation error due to memory allocation)
 - 2. (Optional) Implement the LeNet5.cpp on the Zybo board (Note, the BRAM size exceeds the ZYBO Z-10)
- 2. (Mandatory) Do HLS synthesis for a BiQuad filter source code (See,

https://github.com/HirokiNakahara/FPGA_lecture/blob/master/ BiQuad_Filter/BiQuad_Filter.cpp)

- 1. (Optional) Write a testbench for a BiQuad filter
- 2. (Optional) Implement a BiQuad filter on the Zybo board
- 3. (Optional) Optimize a BiQuad filter
- 3. (Mandatory) Report differences between Xilinx and Intel FPGAs

Final Report Submission

- Submit an e-mail attaching a PDF including an exercises 13
- Also, back your Zybo board to Nakahara Lab.
 - Note that, return to my office administrator (Ms. Shimura)
 - Room#: S3-402
 - She works from AM10:00 to PM16:00 except for Wednesday

My e-mail address: nakahara@ict.e.titech.ac.jp

Deadline is 31st, Aug., 2018