ATLANTIS PRESS
Advances in Engineering Research, volume 118
2nd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2017)

A Visual Tracking Deep Convolutional Neural Network Accelerator

Zhiyong Qin[1,a], Lixin Yu[1,b]

[1]Beijing Microelectronic Technology Institute, Beijing 100076, China.

[a]qinzy2010@163.com, [b]yulx2013@126.com

Keywords: Convolutional neural network, visual tracking, accelerator.

Abstract. Convolutional Neural Network (CNN) has been widely used in computer vision problems including image classification, object detection and tracking and semantic segmentation which has obtained state-of-art results. However, CNN is both computation and bandwidth intensive. At present, we usually train and use CNN on CPU, GPU and even GPU clusters which are impractical for embedded platforms. Therefore, we need design CNN accelerator for low power, high computation and high bandwidth processing. There are already some accelerator which is mainly accelerate 2-D convolution, but not high energy-efficiency for 3-D convolution. So we propose an architecture which is effective for large depth 3-D convolution and we support three different scales kernel size. We use a visual tracking CNN algorithm-MDNet to verify our architecture and make no accuracy loss. We evaluate our system using 65nm library which has a small footprint of 2.58 mm^2 and 224 mW.

Introduction

Convolutional Neural Network is a well-known deep learning technology which has been used in many computer vision tasks such as video surveillance and mobile robot vision and obtains state-of-the-art results. Visual tracking is a basic and old problem in computer vision. In VOT 2015[1], a visual tracking challenge, a CNN-based method achieve the champion.

CNN has achieved so good results and provoked the attention of academia and industry, but it demands much computation and memory resources. And the CNN-based method will generate many intermediate values which aggravates bandwidth pressure. Therefore, people have to use many CPUs and GPUs with large cache and DDR. But this is impractical for embedded platforms which is limited in power and resource.

To solve the problem, many researchers have designed various accelerators which focus on computation and memory. HWCE is a many-core coprocessor which is state-of-the-art at energy-efficiency. But its memory reuse system is not appropriate for 3-D convolution which need cache too much image data for reusing. Origami accelerate a scene labeling algorithm and can achieve a high power efficiency while it need cache a block of data whose size is the length of image multiply the width and depth of kernel. When the depth of a kernel is too large and the image length is not too small, its SRAM is not sufficient to cache the block of data.

In this paper, we firstly analyze the architecture and configuration of a CNN and draw a conclusion that convolution layer need more computation resource and fully-connected layer need more memory resource. Then we propose a CNN accelerator architecture which can support large depth 3-D convolutions. And we support three different 2-D convolution kernels which respectively are 7×7, 5×5 and 3×3. Finally, we synthesized our architecture using 65nm standard VT library and our implementation is in a small footprint of 2.58 mm^2 and 224 mW.

Related Work

Convolutional Neural Networks have been implemented on various platforms, including CPU, GPU, FPGA and ASIC. There are many libraries such as caffe of UC Berkerly [2], tensorflow of Google [3], neon of Nervana Systems [4] and so on which are very fast for forward and backward propagation based on CPU and GPU. But the energy-efficiency of CPU and GPU is low due to their flexibility. There are also many implementations of FPGA and ASIC.

CNP [5] is the almost earliest FPGA-based processor for convolutional networks. Its peak performance is 9.8 billion connections per second and 4 billion connections per second on average at up to 200MHz using 18 bit fixed-point data. It is implemented on a Spartan 3A DSP 3400 FPGA with a peak power consumption of 15W. It implements nearly all the convolutional network operations including 2-D convolutions, spatial pooling and non-linear functions. NeuFlow [6] is a scalable architecture which can process multi-layer network and it also features a dataflow compiler that transform a high-level flow-graph representation into machine code of neuFlow. It can achieve a real performance of 147 GOP/s on a Xilinx Virtex 6 FPGA with a power consumption of 10W. It optimize the grid of filter-based systems that different process element handle different operations. Chen Zhang etc. of Peking University implements a FPGA-based accelerator on a VC707 board [7]. Their implementation achieves a peak performance of 61.62 GFLOPS under 100MHz working frequency with power consumption of 18.61W. They make model of CNN on 32 bits float-point data and based on HLS which results in low energy-efficiency. Jiantao Qiu etc. of Tsinghua University implement a system on Xilinx Zynq ZC706 board which achieves performance of 137.0GOP/s under 150MHz working frequency[8]. It consumes 9.63W using 16-bit quantization. Because it aims to process large CNN model such as ZF, VGG and so on whose mode size are several millions parameters, they mainly deal with the model compression and data quantization problems. But the accelerator only supports 3*3 convolution which makes it only used in a small cases.

DianNao [9] series are all ASIC accelerators which accelerate machine learning including CNN and DNN. DianNao is a three-stage pipeline architecture with input, output and weights memory. It performs 452GOP/s in a small footprint of 3.02 mm^2 and with a power consumption of 485mW at 65nm. DaDianNao [10] is a custom multi-chip machine-learning architecture which reduce energy by up to 150x using 64 nodes. HWCE [11] is a coprocessor for accelerating convolution which used in a many-core cluster. It use a data stream to input image data so it can reduce the wait time of convolution engine. And it can achieve an amazing performance of 2750GOPS/W using ST FDSOI 28nm at voltage of 0.4V. But it doesn't appropriate for 3-D convolutions, although it can add different feature maps results, if the feature map is very deep it need many buffer to cache the intermediate data and make the control more complex.

Primer on Convolutional Neural Network

Basic Concept of CNN.

CNN usually consists of a series of continuous layers, including the input layer, the output layer and a number of hidden layers. CNN with more than two hidden layers can be considered as deep neural network. The main type of hidden layers are convolution, and fully-connected.

Convolutional layer gets input from previous layer and convolves with convolutional kernels to obtain the output feature maps. In this paper, 'in' denotes the input of convolutional layer, 'o' denotes the output feature maps, 'i', 'j', 'k' separately denotes the index of three dimensions, 'w' denotes the convolutional kernels, 'x', 'y', 'z' separately denotes the three dimensions of kernels, 'k' denotes the number of feature maps, 's' denotes the stride of kernel and 'b' denotes the offset of feature map, so the calculation of the convolution layer is as follows:

$$o(i,j,k) = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}\sum_{z=0}^{Z-1} w^k(x,y,z)\cdot in(x+i\times s, y+j\times s, z) + b^k \quad (1)$$

where X, Y, and Z denote the size of the corresponding dimension of the convolution kernel. Each output point is calculated by the convolution operation of the convolution kernel and the input part. A convolutional kernel moves from left to right, from top to bottom and point by point in a fixed step. All the convolutional kernels take the same operation to get a series of output feature maps. The relationship of output dimension and input dimension are as follows:

$$(O_x, O_y, O_z) = \left(\frac{I_x - X}{s} + 1, \frac{I_y - Y}{s} + 1, k\right) \quad (2)$$

where I_x and I_y respectively denotes the dimension of input feature map, O_x,O_y and O_z denotes the three dimension of output feature map apart. The output feature map needs to be processed by the activation functions. Most people use ReLU as activation functions which can be

expressed with:

$$ReLU(x) = max(0, x) \quad (3)$$

From the equation we can know that it is very computationally efficient and converges much faster.

Fully Connected Layer is a linear mapping from the input to the output. The expression is as follows:

$$O = W \times IN + B \quad (5)$$

Where 'O' denotes the output feature vector, 'IN' denotes the input feature vector, 'W' denotes the weights matrix and 'B' denotes the bias vector.

A Visual Tracking CNN.

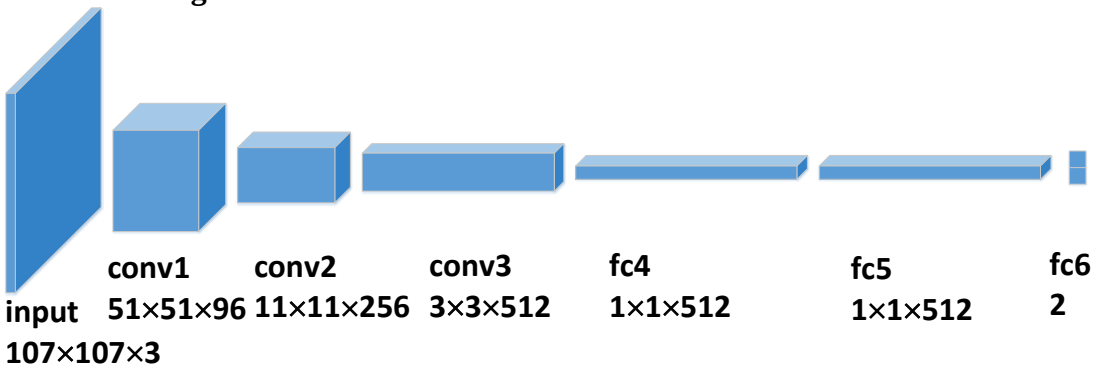

Fig. 1 a visual tracking CNN that won the 2015 VOT challenge

Figure 1 shows a visual tracking CNN, taken from [12]. This CNN is composed of 6 layers. The first 3 layers are convolutional layers and layers 4~6 form a fully connected network. The algorithm receives 107×107 RGB input images that are cropped from images whose largest size is 1280×720. As is shown in Figure 1, first layer receives 3 input feature maps in 107×107 resolution and then output 96 feature maps in 51×51 resolution. The first layer kernel size is 7×7×3 and the sliding stride is 2 pixels. The other layers is similar to the first layer and the parameter is showed as table 1.

Table 1 MDNet Configuration (fm: feature maps)

Layer	1	2	3	4	5	6
input_fm	3	96	256	512	512	512
output_fm	96	256	512	512	512	512
fm length	51	11	3	1	1	1
kernel	7×7×3	5×5×96	3×3×256	3×3×512	1×1×512	1×1×512
stride	2	2	1	#	#	#

Architecture

System Overview.

Figure 2 shows a top-level diagram of our implementation. In our architecture, we firstly read data into image buffer. We usually read one frame image, if the image is too large we crop it and read it many times. Then we read weight and bias data into weight and bias buffer. We read weight data as many as possible so long as it does not exceed the capacity of buffer. When the buffer cannot accommodate all the weight data, we load the weight data many times for an image, in this way we can ensure every picture data be loaded only once. This can lighten the bandwidth pressure. At the same time, the control unit give control signals to image, weight and bias address generator to manage the dataflow. When all the data is ready, the control unit give computing unit enable signals to start computing process and give a valid signal to demonstrate whether the output data is valid.

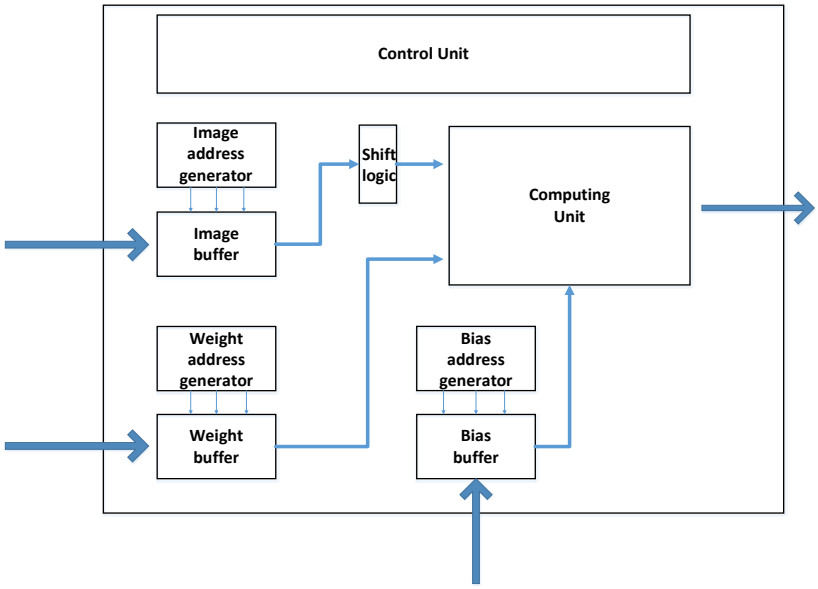

Fig. 2 Top-level diagram of our architecture for selected parameters

Memory System.

1) Image Buffer and Address Generator

Image Buffer stores input image data and provide data for computing unit. To provide the maximum internal bandwidth, we divide the image buffer into 150 banks which is consistent with our computing unit. For every bank we want to keep the image buffer as small as possible and at the same time keep the bandwidth balanced. So we select a largest crop of image whose size is 400 for a bank. We store the image data in terms of block and every block do not exceed 150. Figure 3 shows we how to split a picture into 150 banks. According to the size of weight kernel, we use buffer from bank1 to bank150 to store a 5×5×6 block every time, so the whole image is divided into 5×5×16 blocks. When the weight kernel is not equal to 150, we cut it to a value as much as possible which is less than 150. For example, the kernel size of layer1 is 7×7×3, so we use buffer from bank1 to bank147 to store image data, and leave the bank148 to bank150 not used.

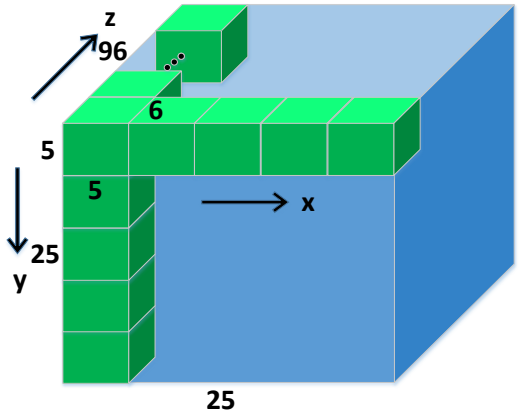

Fig. 3 an example of image data and the organization of blocks

The image address generator produce the read and write address for input image data and the data provided for computing unit. Actually the image address generator is a finite state machine (FSM). For writing address generation, we only need increase the writing address one every time. For reading address generation, there are four main states which separately are initial state, z-direction state, x-direction state, y-direction state. The initial state is the beginning of reading process. In the z-direction state, all the reading address accumulates each time. In the x-direction state, the reading address of black region need to be accumulated and the other banks address need to be restored. The y-direction state is similar to x-direction state, the difference is the reading address need to be accumulated is different.

2) Weight and Bias Buffer and Address Generator

Weight buffer is used to store weight data and provide data for computing unit. Weight buffer is designed mainly according to the size of weight kernel. We select an appropriate 3-D kernel size which is 150 to accelerate the computing unit and balance bandwidth problem. We divide the weight buffer into 150 banks to store weight data separately. Bias buffer is so small in terms of weight buffer, so we design the size of bias buffer as 512 which is the largest size of bias values. Due to image is large in terms of weight, so a weight and bias data will be reused many times. Therefore, the weight and bias address generation can increase, decrease or hold the reading or writing address.

3) Computing Unit and Shift Logic

The architecture of computing unit is shown as figure 4. It mainly consists of multiplier array, adder trees and accumulator and relu unit which is a 3-staged pipelined architecture. The input of computing unit is 16-bit fixed-point data and the point can float within limits. The multiplier array includes 150 multipliers which can process 150 16×16 fixed-point multiplication every cycle. The adder tree can process 150 additions simultaneously. The accumulator is used to process very deep feature map kernels which can cache the intermediate result and accumulate the intermediate data to get the final results. The Relu unit is to process ReLu operation. The Shift Logic is used to process the sliding window operation of convolution. It is implemented by a multiplexer array. We generate selecting signal by the stride of kernel and our architecture supply 7×7, 5×5 and 3×3 kernels.

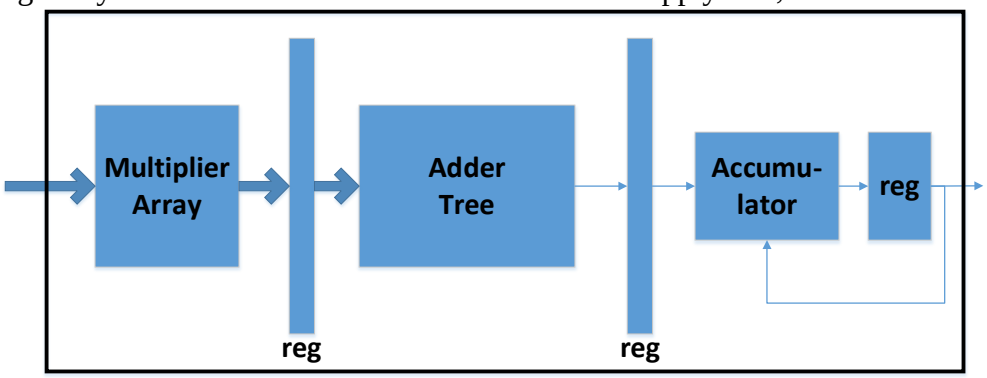

Fig. 4 computing unit architecture

4) Control Unit

The control unit dominate all the data flow. The time diagram is shown as figure 5.

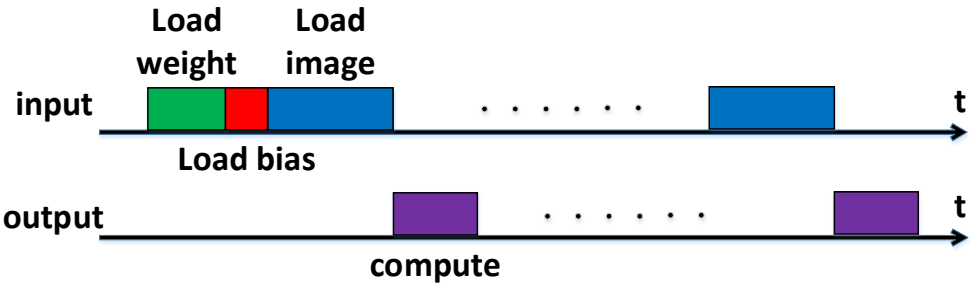

Fig. 5 Time diagram of input and output data

We firstly give weight address generator start signal which make it working. Then we select weight buffer to load all the weight data. After that, we write bias buffer and image buffer. When all the data is ready, we enable computing unit to calculate and enable the output. When a frame picture has been processed, we continue to read image data from external and calculate the result after data is ready.

Results

We use MDNet as our algorithm prototype and the MATLAB model is served as the specification for the Verilog implementation. The software setup environments are MATLAB R2016a, matconvnet 1.23 and CUDA 8.0. We test the MATLAB and Verilog model using OTB100

dataset [13] which is an object tracking benchmark containing 100 sequences. The result is shown as figure 6, we can know that the area under curve (AUC) of our RTL model has no accuracy loss and even a little improved compared to original MATLAB algorithm model. While the non-zero value of the output of Verilog model is a little less than the original model. It is because our architecture is not so robust for recovering when it lose the target. For the bird1 sequence, after frame 144, the algorithm lose the target lasting for the end while the original algorithm can find the target at the last frames.

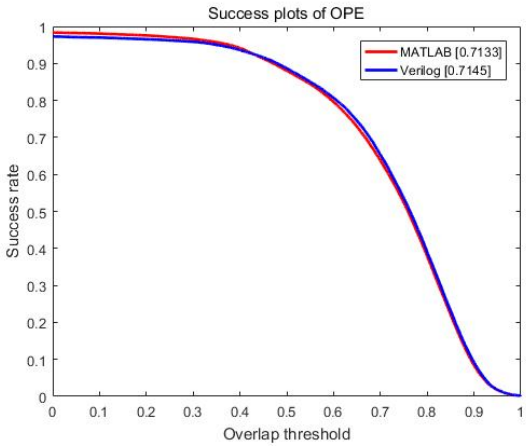

Fig. 6 Area under curve of original MATLAB algorithm and our verilog model

Finally we use Synopsys Design Compiler to evaluate our Verilog implementation. We synthesize our Verilog model using the SMIC 65nm standard VT library. The area of our implementation is 2.58 mm^2 and the power is 224 mW.

Conclusion

We use deep visual tracking algorithm as the entry point of deep convolutional neural network. The state-of-the-art CNN algorithm mostly use 3-D convolution and the feature maps are usually large, but at present many accelerator optimize their architecture mainly including the memory system for 2-D convolution. We propose an architecture which can process large depth 3-D convolution kernels. And our implementation has no accuracy loss which is in a footprint of 2.58 mm^2 and 224 mW.

Reference

[1] http://www.votchallenge.net/vot2015/

[2] Y. Jia, “Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding,” 2013. http://caffe.berkeleyvision.org

[3] http://www.tensorflow.org/

[4] Nervana Systems Inc., “Neon Framework,” 2015. http://neon.nervanasys.com

[5] Farabet, Clément, et al. "Cnp: An fpga-based processor for convolutional networks." 2009 International Conference on Field Programmable Logic and Applications. IEEE, 2009.

[6] Farabet, Clément, et al. "Neuflow: A runtime reconfigurable dataflow processor for vision." Cvpr 2011 Workshops. IEEE, 2011.

[7] Zhang, Chen, et al. "Optimizing fpga-based accelerator design for deep convolutional neural networks." Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2015.

[8] Qiu, Jiantao, et al. "Going deeper with embedded fpga platform for convolutional neural network." Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable

Gate Arrays. ACM, 2016.

[9] Chen, Tianshi, et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning." ACM Sigplan Notices. Vol. 49. No. 4. ACM, 2014.

[10] Chen, Yunji, et al. "Dadiannao: A machine-learning supercomputer." Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2014.

[11] Conti, Francesco, and Luca Benini. "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters." Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium, 2015.

[12] Nam, Hyeonseob, and Bohyung Han. "Learning multi-domain convolutional neural networks for visual tracking." arXiv preprint arXiv:1510.07945 (2015).

[13] Wu, Yi, Jongwoo Lim, and Ming-Hsuan Yang. "Object tracking benchmark." IEEE Transactions on Pattern Analysis and Machine Intelligence 37.9 (2015): 1834-1848.