[DL] A Survey of FPGA Based Neural Network Accelerator

KAIYUAN GUO, SHULIN ZENG, JINCHENG YU, YU WANG AND HUAZHONG YANG, Tsinghua University, China

Recent researches on neural network have shown great improvement in computer vision over traditional algorithms based on handcrafted features and models. Neural network is now greatly adopted in regions like image, speech and video recognition. But the great computation and storage complexity of neural network based algorithms poses great difficulty on its application. CPU platforms are hard to offer enough computation capacity. While GPU platforms are highly parallelized, the energy efficieny is low. The high energy cost of GPU causes problems for a wide application of neural network.

To address the above problems, various FPGA based hardware accelerators for neural networks have been proposed. Specialized hardware are designed to achieve high speed and low power neural network process. In this paper, we give an overview of previous work on neural network accelerators based on FPGA and summarize the main techniques used. Investigation from software to hardware, from circuit level to system level is carried out to complete analysis of FPGA based neural network accelerator design and serves as a guide to future work.

Additional Key Words and Phrases: FPGA, Neural Network

ACM Reference Format:
Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang AND Huazhong Yang. 2017. [DL] A Survey of FPGA Based Neural Network Accelerator. *ACM Trans. Reconfig. Technol. Syst.* 9, 4, Article 11 (December 2017), 7 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Recent research on Neural Network (NN) is showing great improvement over traditional algorithms in computer vision. Various network models, like convolutional neural network (CNN), recurrent neural network (RNN), have been proposed for image, video, and speech process. CNN [12] improves the top-5 image classification accuracy on ImageNet [19] dataset from 73.8% to 84.7% and further helps improve object detection [3] with its outstanding ability in feature extraction. RNN [8] achieves state-of-the-art word error rate on speech recognition. In general, NN features a high fitting ability to a wide range of pattern recognition problems. This makes NN a promising candidate to many artificial intelligence applications.

But the computation and storage complexity of NN models are high. The research on NN is also increasing the size of NN models. The largest neural network model for an 224×224 image classification requires upto 39 billion floating point operations (FLOP) and more than 500MB model parameters [20]. As the computation complexity is propotional to the input image size, processing images with higher resolutions may need more than 100 billion operations.

Traditional hardware platforms are not suitable for neural network process. A common CPU can perform 10-100G FLOP per second, and the power efficieny is usually below 1GOPs/W. So CPUs neither meet the high performance requirements in cloud applications nor the low power

Author's address: Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang AND Huazhong Yang, Tsinghua University, Tsinghua University, Beijing, Beijing, 100084, China, gky15@mails.tsinghua.edu.cn,yu-wang@mail.tsinghua.edu.cn.

1936-7406/2017/12-ART11 $15.00
https://doi.org/0000001.0000001

requiremetns in mobile applications. In contrast, GPUs offer upto 10TOP/s peak performance and is a good choice for high performance neural network applications. Development frameworks like Caffe [11] and Tensorflow [1] also offers easy-to-use interfaces which makes GPU the first choice of neural network acceleration. But GPUs are power consuming and thus not suitable for mobile applications.

On the other hand, FPGA is becoming a candidate to implement energy efficient neural network accelerator. With a specific hardware design, FPGAs are able to implement high parallelism and make use of the properties of neural network computation to remove unecessary logic. Therefore FPGAs are possible to achieve higher energy efficieny compared with CPU and GPU.

But FPGA based accelerator designs are still faced with two problems:

- Current FPGAs usually support working frequency at 100-300MHz, which is much less than CPU and GPU. The FPGA's logic overhead for reconfigurability also reduces the overall system performance. Straight forward design on FPGA is hard to achieve high performance and high energy efficiency.
- Implementation of neural networks on FPGAs is much harder than that on CPUs or GPUs. Development framework like Caffe and Tensorflow for CPU and GPU is needed for FPGA.

Many researches on the above two problems have been carried out for energy efficient and flexible FPGA based neural network accelerator. In this paper, we summarize the techniques proposed in these work. Specifically, we will introduce the techiques from the following aspects:

- We investigate current techniques for high performance and energy efficient neural network accelerator designs. Techniques in both software level and hardware level are evaluated.
- We investigate state-of-the-art automatic design methods of FPGA based neural network accelerators.

The rest part of this paper is organized as follows:

2 PRELIMINARY ON NEURAL NETWORK

In this section, we introduce the basic operations included in neural network algorithms. Neural network is a bio-inspired model, which usually includes several layers. Each layer receives input from a set of neurons and output a set of neurons. The synapses connecting input and output neurons are modeled as parameters, which is referred to as weights in this paper. In the rest part of this section, we introduce different types of layers in neural network models.

Fully conencted (FC) layer implements a connection between every input neuron and output neuron with a weight. This type of layer is adopted in both CNN and RNN. The input and output neurons of an FC layer are two vectors $\mathbf{x}$ and $\mathbf{y}$. The weights of this layer can be modeled as a matrix W. A bias vector b is added to each of the output neuron. The computation of this layer is described as equation 1.

$$\mathbf{x} = W\mathbf{y} + \mathbf{b} \tag{1}$$

Convolution (CONV) layer is used for 2-d neuron process. This is commonly adopted in CNN for image process. The input and output neurons of this layer can be described as sets of 2-d feature maps, F_{in} and F_{out}. Each feature map is denoted as a channel. A CONV layer implements a 2-d convolution kernel K_{ij} for each input and output channel pair and a bias scalar b_i for each output channel. The computation of a CONV layer with M input channels and N output channels can be described as equation 2.

$$F_{out}(j) = \sum_{i=0}^{M-1} \text{conv2d}(F_{in}(i), K_{ij}) + b_j,\ j = 0, 1, ..., N-1 \tag{2}$$

There are varieties of 2-d convolutions in CONV layer. Usually standard convolution with padding is used when the kernel size is 3×3. For larger kernels like 5×5 and 7×7, a stride larger than 1 is usually used to reduce the number of operation. Recent work is also using 1×1 convolution kernels [9, 10].

Non-linear layer applies a non-linear function on each of the input neurons. Sigmoid function and tanh function are commonly adopted in early models are are still used in RNN for acoustic or speech recognition. Rectified linear unit (ReLU) [12] is the adopted in many state-of-the-art models. This function maintains the positive neurons and filters negative neurons as zero. Varieties of ReLU are also used, such as PReLU and Leaky ReLU [22].

Pooling layer is also used for 2-d neuron process like CONV layer. A pooling layer downsamples each of the input channel respectively, which helps reduce feature dimension. There are two kinds of down sampling method: average pooling and max pooling. Average pooling splits a feature map into small windows, i.e. 2×2 windows, and finds the average value of each window. Max pooling method finds the maximum value in each window. Common window size includes 2×2, stride=2 and 3×3, stride=2.

Element-wize layer is usually used in RNN and is introduced in ResNet [9]. This layer receives two neuron vectors of the same size and applies element-wise operations on corresponding neurons of the two vectors. In ResNet, this layer is element-wise addition. For RNN, this layer can be element-wise subtraction or multiplication.

Among these types of layers, FC layer and CONV layer contributes to most of the computation and storage in neural networks. In the following sections, both software level and hardware level designs focus on these two types of layers.

3 DESIGN METHODOLOGY

Before going into the details of the techniques used for fast and energy efficient neural network accelerator, we first give an overview of the design methodology. In general, the design target of a neural network processing system includes the following three aspects: high model accuracy, high throughput and high energy efficiency. For certain applications, high flexibility should also be considered.

In general, a larger neural network model usually results in a higher model accuracy. Different network structures like the ones in [9, 12, 20] surely affect model accuracy, but is out of the discussion of this paper. With a same model, applying model compression methods can acheive the trade-off between throughput and model accuracy. Some of the model compression methods even achieves acceleration without model accuracy loss.

The throughput of a neural network processing system can be expressed by equation 3. With model compression methods, we can reduce the workload. With a certain FPGA chip, the on-chip resource is limited. Increasing the peak performance means to reduce the size of each computation unit and increase the working frequency. Reducing the size of computation units usually means to simplify the basic operations in neural network model, which is a hardware-software co-design problem. Increasing working frequency, on the other hand is pure hardware design work. A high utilization ratio is kept by reasonable parallelism implementation and efficient memory system. Most of this part is affected by hardware design. But model compression can also reduce the storage requirment of a neural network model and benefits the memory system.

$$throughput = \frac{peak\_performance \times utilization}{workload} \tag{3}$$

Energy efficiency is evaluated by the number of operations (multiplication or addition in this case) executed with unit energy cost. Given a certain network model, the energy efficiency of a

neural network processing system is inversely proportional to the energy cost, which is expressed in equation 4. The energy cost comes from 2 parts: computation and memory access.

$$E_{total} = N_{effect\_op} \times E_{unit\_op} + N_{mem\_access} \times E_{unit\_mem\_access} \quad (4)$$

The first item in equation 4 is the energy cost for computation. This part is greatly affected by model compression. Model compression methods can reduce the actual number of operations carried out on hardware, N_{effect_op} and simplify the operations to reduce the unit energy cost of a single operation E_{unit_op}. Given an FPGA chip, E_{unit_op} is also affected by its hardware implementation. The second item in equation 4 is the energy cost for memory access. The number of memory access N_{mem_access} is affected by the memory system and scheduling method. The energy for each memory access can be reduced by model compression methods by reducing the bit-width of data.

From the analysis of throughput and energy, we see that neural network accelerator involves software-hardware co-design. In the following sections, we will introduce previous work in both software and hardware level.

4 SOFTWARE DESIGN: MODEL COMPRESSION

As introduced in section 3, the design of energy efficient and high performacne neural network accelerator involves software and hardware co-design. In this section, we investigate the software level network model compression methods. Many researches on this topic have been proposed to reduce the number of weights or reduce the number of bitwidth for the neurons and weights, which helps reudce the computation and storage complexity. But these methods can also sacrifice the model accuracy. The trade-off between model compression and model accuracy loss is discussed in this section.

4.1 Data Quantization

One of the most commonly used method for model compression is the quantization on the weights and neurons. The neurons and weights of a neural network is usually represented by floating point data in common developing frameworks. Recent work try to replace this representation with low-bit fixed-point data or even a small set of trained values. On one hand, using less bits for each neuron or weight helps reduce the bandwidth and storage requirement of the neural network processing system. On the other hand, using a simplified representation reduce the hardware cost for each operation. The benefit on hardware will be discussed in detail in section 5. Two kinds of quantization methods are discussed in this section: linear quantization and non-linear quantization.

4.1.1 Linear Quantization. Linear quantization finds the nearest fixed-point representation of each weight and neuron. The problem of this method is that the dynamic range of floating point data greatly exceeds that for fixed point data. Most of the weights and neurons will suffer from overflow or underflow. Qiu, et al. [18] finds that the dynamic range of the weights and neurons in a single layer is much more limited and differs across different layers. Therefore they assign different fractional bit-widths to the weights and neurons in different layers. To decide the fractional bit-width of a set of data, i.e. the neurons or weights of a layer, the data distribution is first analyzed. A set of possible fractional bit-widths are chosen as candidate solutions. Then the solution with the best model performance on training data set is chosen. In [18], the optimized solution of a network is chosen layer by layer to avoid an exponential design space exploration. Guo, et al. [5] further improves this method by fine tuning the model after the fraction bit-width of all the layers are fixed.

The method of choosing certain fractional bit-width equals to scale the data with a scaling factor of 2^k. Li, et al. [13] scales the weights with trained parameter W^l for each layer and quantize the

weights with 2-bit data, representing W^l, 0 and $-W^l$. The neurons in this work is not quantized. So the the network still implements 32-bit floating point operations. Zhou, et al. [25] further quantize the weights of a layer with only 1 bit to $\pm s$, where $s = E(|w^l|)$ is the expectation of the absolute value of the weights of this layer. Linear quantization is also applied to the neurons in this work.

4.1.2 Non-linear Quantization. Compared with linear quantization, non-linear quantization independently assigns values to different binary code. The translation from a non-linear quantized code to its corresponding value is thus a look-up table. This kind of methods helps further reduce the bit-width used for each neuron or weight. Chen, et al. [2] assign each of the weight to an item in the look-up table by a pre-defined hash function and train the values in look-up table. Han et al. [7] assigns the values in look-up table to the weights by clustering the weights of a trained model. Each look-up table value is set as the cluster center and further fine-tuned with training data set. This method is able to compress the weights of state-of-the-art CNN models to 4-bit without accuracy loss. Zhu, et al. [26] propose the ternary quantized network where all the weights of a layer are quantized to three values: W^n, 0, and W^p. Both the quantized value and the correspondance between weights and look-up table are trained. This method sacrifices less than 2% accuracy loss on ImageNet data set on state-of-the-art network models. The weight bit-width is reduced from 32-bit to 2-bit, which means about 16× model size compression.

4.1.3 Comparison. Experimental results of this method on VGG-16 model [20] are shown in Figure **??** according to [18] and [5]. 16-bit fixed point data format shows a similar performance to the 32-bit floating point baseline. By introducing different fractional bit-widths for different layers, 8-bit fixed point format also introduces negligible accuracy loss. Further narrowing the bit-width to 6 causes an obvious accuracy loss.

4.2 Weight Reduction

Besides narrowing the bit-width of neurons and weights, another method for model compression is to reduce the number of weights. One kind of method is to approximate the weight matrix with a low rank representation. Qiu, et al. [18] compress the weight matrix W of an FC layer with singular value decomposition. An $m \times n$ weight matrix W is replaced by the multiplication of two matrices $A_{m\times p}B_{p\times n}$. For a sufficiently small p, the total number of weights is reduced. This work compress the largest FC layer of VGG network to 36% of its original size with 0.04% classification accuracy degradation. Zhang, et al. [24] use similar method for convolution layers and takes the effect of the following non-linear layer into the decomposition optimization process. The proposed method achieves 4× speed up on state-of-the-art CNN model targeting at ImageNet, with only 0.9% accuracy loss.

Pruning is another kind of method to reduce the number of weights. This kind of method directly remove the zeros in weights or remove those with small absolute values. The challenge in pruning is how to make more weights zero while keeping the model accuracy. One solution is the application of lasso object function during training. Liu, et al. [15] apply the spase group-lasso object function on the AlexNet [12] model. 90% weights are removed after training with less than 1% accuracy loss. Another solution is to prune the zero weights during training. Han, et al. [7] directly removes the values in network with zero or small absolute value. The left weights are then fine-tuned are training set to recover accuracy. Experimental result on AlexNet show that 89% weights can be removed while keeping the model accuracy.

5 HARDWARE DESIGN: EFFICIENT ARCHITECTURE

In this section, we investigate the hardware level techniques used in state-of-the-art FPGA based neural network accelerator design to achieve high performance and high energy efficiency. We

classify the techniques into 3 levels: computation unit level, loop mapping level, and memory system level.

5.1 Computation Unit Designs

Computation unit level design affect the peak performance of the neural network accelerator. With a certain FPGA chip, the available resource is limited. A smaller computation unit design means more computation units and higher peak performance. A carefully designed computation unit array can also increase the working frequency of the system and thus improve peak performance.

5.1.1 Low Bit-width Unit. Reduce the number of bit-width for computation is a direct way to reduce the size of computation units. The feasibility of using less bits comes from the quantization methods as introduced in section 4.1. Most of the state-of-the-art FPGA designs replace the 32-bit floating point units with fixed point units. Podili, et al. [17] implements 32-bit fixed point untis for the proposed system. 16-bit fixed point units are widely adopted in [4, 14, 18, 21, 23]. ESE [6] adopts 12-bit fixed-point weight and 16-bit fixed point neurons design. Guo, et al. [5] use 8-bit units for their design on embedded FPGA. Recent work is also focusing on extremely narrow bit-width design. Experiments in [16] shows that FPGA implementation of Binarized Neural Network (BNN) outperforms that on CPU and GPU.

5.1.2 Fast Convolution Unit.

5.1.3 Frequency Optimization Methods.

5.2 Loop Mapping Strategies

5.3 Memory System

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*. 2285–2294.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[4] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates. (2017), 152–159.

[5] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2017. Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2017).

[6] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA.. In *FPGA*. 75–84.

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

[8] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[10] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[13] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[14] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. 2016. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 1–9.

[15] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 806–814.

[16] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 77–84.

[17] Abhinav Podili, Chi Zhang, and Viktor Prasanna. 2017. Fast and efficient implementation of Convolutional Neural Networks on FPGA. In *Application-specific Systems, Architectures and Processors (ASAP), 2017 IEEE 28th International Conference on*. IEEE, 11–18.

[18] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. 2016. 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, 26–35.

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

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

[21] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. 2017. Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 62.

[22] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

[23] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 1–8.

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

[25] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[26] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).