FPGA-based convolutional layer implementation

A K Berzin and E S Dergunov
National Research Tomsk State University, Russia
berzin.ak@ya.ru

Abstract. In this paper, several approaches to the implementation of a convolutional layer of a neural network on FPGAs for use in embedded systems are considered, as well as a number of optimizations necessary to speed up the operation of such a layer. At the end of the work, an FPGA-based implementation is proposed that is comparable in performance with the reference CPU-based one.

1. Introduction

Deep learning is a subset of machine learning focused on the use of artificial neural networks (ANNs). It is one of the fastest growing areas of science and technology in the twenty-first century. Solutions based on the use of artificial neural networks are ahead of traditional algorithms for computer vision [1], speech recognition, natural language processing, and many other areas.

Along with the development of the Internet of Things (IoT), interest in the use of ANNs in embedded systems has increased [2]. Standard solutions based on central processing units (CPUs) and graphics processing units (GPUs), that are usually used for implementation of ANNs, are not appropriate for embedded systems due to strict limitations on power consumption and size of the components used in such systems. Oftentimes, 32-bit ARM processors that are widely used in embedded systems cannot provide enough computing power for using artificial neural networks, which forces one to find other solutions to the problem: completely abandon the use of neural networks, choose simpler but less accurate neural network architectures (for example, SqueezeNet [3]) or use more efficient processors.

The structure of neural networks allows for parallel computation. Essentially, a neural network is a composition of matrix multiplications, matrix convolutions, and element-wise nonlinear functions. Each element in the matrix can be calculated separately from the others, therefore, up to a certain limit, with an increase in the number of threads in which calculations are performed, the time for calculating the result of a matrix operation decreases. Due to this fact, the computation of neural networks on GPUs, specially designed for fast application of matrix operations, is much faster than on CPUs.

FPGAs can be effectively used in parallelizing computations due to their internal structure. Furthermore, these integrated circuits usually have a relatively low power consumption and small size, which is the reason they are often used in embedded systems. Unlike CPUs and GPUs, FPGAs do not have computational cores, and therefore the maximum number of threads in which computations are performed is limited only by the number of logical elements on a chip. These facts caused a growing interest in the use of FPGAs to accelerate neural networks.
2. Convolutional neural networks operations
In a typical convolutional neural network, there are multiple convolutional, pooling/subsampling and normalization layers. The extraction of high-level features happens due to different operations with input data in the layers. These features can be represented as a finite set of output classes categorized by output fully-connected layer.

Figure 1. Example of a single convolutional neural network layer.

Convolution involves 3-dimensional multiply and accumulated operations of input feature maps with convolutional kernels to compute output feature maps. Convolutional can constitute more than 90% of total operations [1][4].

3. Challenges of hardware implementation
Implementation of convolutional neural networks (CNNs) on embedded platforms is a difficult task due to limited power and FPGA resources (such as digital signal processing (DSP) blocks, on-chip memory and logic elements) and high requirements to computational capabilities. On the other hand, there are specific FPGA features that allow them to be more effective than CPUs and GPUs in computations that are related to CNNs. Comparing FPGA with application specific integrated circuits (ASICs) shows that the latter requires longer development time, needs more resources, should be produced in larger volumes in order to be cost effective and is impossible to reconfigure after manufacturing [4].

One of the main strategies to implement algorithms on FPGA is to use parallel computing and pipelining in design. Because convolutional algorithms are easy to parallelize, using FPGAs for CNN inference may be an effective option. Due to the possibility of creating a custom algorithmic conveyor for operators such as an adder tree, FPGA gives a great performance comparable to CPU and GPU especially for larger kernel sizes [5].

Access to external memory resources is one of the main bottlenecks in CNN implementation on CPU and GPU. Unlike them, FPGA has on-chip memory resources and therefore allows to decrease the number of calls to the external memory.

Trying to implement a convolutional neural network as a sequence of layers is a good way to achieve maximum throughput and performance, but the number and size of convolutional layers is usually too large for the amount of logical resources available. Another limitation is the speed of data entry. To work around this limitation, it is best to implement computational units that will share the same data, as well as cache data that will be used multiple times [6].

Floating point numbers are commonly used during CNN training. Using this type of numbers to implement convolutional neural networks on FPGAs can be inefficient due to the heavy use of logic cells and DSP blocks. In addition, the synthesis of architectures using floating point arithmetic can take longer. One of the methods frequently used to facilitate computation of the outputs of an artificial neural network is quantization of parameters of the network [2]. These parameters, represented by floating point numbers, are converted into integers with a lower bit depth by a special conversion. The
most common one is to convert the parameters from 32-bit floating point numbers to 8-bit unsigned integers.

4. Hardware design implementation
In the presented work a custom FPGA firmware was created in order to accelerate convolutional operations. The aimed chip is Cyclone V which is usually used in embedded systems.

The CNN acceleration module consists of four blocks: scheduler module, computational module and input and output buffers module. The scheduler module is used for controlling dataflow from input buffer to output, and setting operating mode for computational engines. Computational engine is used for implementing convolutional operation of multiplication and accumulation of input data.

![Figure 2. Proposed design for FPGA-based CNN accelerator.](image)

The convolutional module consists of four modules that implements pipeline structure for data processing. In the first module multiplication of input data on kernel weights is performed. It is followed by an optional requantization (normalization) module. The third module is an adder tree. The main feature of this adder tree is the reuse of adders used in the implementation of convolutions with small kernels, in the implementation of convolutions with large ones for higher utilization of logic resources. The developed module can be used for implementing convolutional operations with kernels with sizes from 2x2 to 7x7. That gives a possibility for acceleration of many popular CNN models.

The synthesis results for the module shows that it consumes less resources than planned and has potential for use in smaller systems. For achieving better speed performance can be used parallel implementation of several modules that can speed up data processing. For aimed chip logic utilization results presented in table 1. From that table can be seen that with 6 modules there is possibility to use other blocks in the programmable logic part of SoC in case if they use low number of RAM and DSP blocks.

| Resource         | 1 acceleration module | 6 acceleration modules |
|------------------|-----------------------|------------------------|
| ALM              | 9854 (9%)             | 62380 (57%)            |
| RAM, kilobits    | 714 (59%)             | 821 (69%)              |
| DSP              | 50 (15%)              | 300 (90%)              |
| \(F_{\text{max}}, \text{MHz}\) | 101                  | 91                     |
5. Conclusion
An FPGA-based implementation of a convolutional neural network layer was proposed. Several techniques were used to effectively speed up computations, such as quantizing weights and input parameters of a layer, using an adder tree instead of a single adder with an accumulator. A number of performance tests were performed on the proposed design. These tests have shown that the implementation described in this article is comparable and even faster than the CPU-based implementation of convolutional layer. In future work more flexible configuration of multiply and adder tree modules is going to be explored and comparison to other hardware embedded platforms will be performed.

References
[1] Krizhevsky A, Sutskever I and Hinton G E 2017 ImageNet classification with deep convolutional neural networks Communications of the ACM. 60(6) pp 84–90
[2] Roth W, Schindler G, Zohrer M, Pfeifenberger L, Peharz R, Tschiatschek S, Froning H, Pernkopf F and Ghahramani Z 2020 Resource-Efficient Neural Networks for Embedded Systems ArXiv abs/2001.03048
[3] Forrest I N, Moskewicz M W, Ashraf K, Han S, Dally W and Keutzer K 2017 SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size ArXiv abs/1602.07360
[4] Yu X, Wang Y, Miao J, Wu E, Zhang H, Meng Y, Zhang B, Min B, Chen D and Gao J 2019 A data-center fpga acceleration platform for convolutional neural networks 29th International Conference on Field Programmable Logic and Applications (FPL) pp 151–158
[5] Zhang C, Li P, Sun G and Cong J 2015 Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays pp 161–170
[6] Suda N, Chandra V, Dasika G, Mohanty A, Ma Y, Vrudhula S, Seo J and Cao Y 2016 Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays pp 16–25