An Overview of FPGA Based Deep Learning Accelerators: Challenges and Opportunities

Teng Wang
University of Science and
Technology of China
School of Software Engineering
Suzhou, China
Email: sa517368@email.ustc.edu.cn

Chao Wang
and Xuehai Zhou
University of Science and
Technology of China
School of Computer Science and Technology
Suzhou, China
Email: cswang@ustc.edu.cn

Huaping Chen
University of Science and
Technology of China
School of Software Engineering
Suzhou, China
Email: hpchen@ustc.edu.cn

Abstract—With the rapid development of in-depth learning, neural network and deep learning algorithms have been widely used in various fields, e.g., image, video and voice processing. However, the neural network model is getting larger and larger, which is expressed in the calculation of model parameters. Although a wealth of existing efforts on GPU platforms currently used by researchers for improving computing performance, dedicated hardware solutions are essential and emerging to provide advantages over pure software solutions. In this paper, we systematically investigate the neural network accelerator based on Field-Programmable Gate Array(FPGA). Specifically, we respectively review the accelerators designed for specific problems, specific algorithms, algorithm features, and general templates. We also compared the design and implementation of the accelerator based on FPGA under different devices and network models and compared it with the versions of CPU and GPU. Finally, we present to discuss the advantages and disadvantages of accelerators on FPGA platforms and to further explore the opportunities for future research.

Index Terms—FPGA, Accelerator, Deep Learning, Neural Network.

I. INTRODUCTION

IN recent years, the research of neural networks (NNs) has been dramatically improved compared with traditional algorithms in the various fields. Various network models, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), have been proposed for image, video and speech processing research domains. Well-trained CNN models have increased the classification accuracy of the top 5 images on the ImageNet data-set from 73.8% to 84.7%, and further improved object detection with its excellent feature extraction capabilities. RNN implements the latest word error rate in speech recognition. In general, due to a high ability of neural networks to fit a wide range of pattern recognition problems, it makes neural networks become a promising candidate for many artificial intelligence applications. However, the neural network models are still suffering from a high computational and storage complexity. In the meantime, the researches on neural networks is still focusing on the boost of the scale of neural network models by now. For example, the largest CNN model for 224x224 image classification requires up to 39 billion floating point operations (FLOP) and over 500 MB model parameters [1]. Since the computational complexity is directly proportional to the size of the input image, processing images with higher resolutions may require more than 100 billion operations.

Therefore, it is particularly important to choose a moderate computing platform for neural network applications. In general, CPU can perform 10-100 GFLOP per second, but power efficiency is usually less than 1 GOP/J. As a consequence it is difficult to achieve the high-performance requirements of cloud applications and the low power requirements of mobile apps. In contrast, GPUs offer the peak performance of up to 10 TOP/s, therefore it is an excellent choice for high-performance neural networking applications. In addition to CPU and GPU, FPGA is gradually becoming a candidate platform for energy-efficient neural network processing[2]. FPGAs can achieve high parallelism and simplify logic according to the calculation process of a neural network with the hardware design for specific models. Some researches show that the neural network model can be simplified in a hardware-friendly manner without affecting the accuracy of the model. Therefore, FPGA can achieve higher energy efficiency than CPU and GPU[3].As shown in Figure 1, until last year, the number of FPGA-based neural network accelerators published in the IEEE eXplore has reached 69 and is still on the rise. It is enough to illustrate the research trend in this direction.

II. BACKGROUND

Deep learning combines low-level features to form more abstract high-level representation attribute categories or features to discover distributed feature representations of data.Its concept was proposed by Hinton et al. in 2006[4]. Based on Deep Belief Network (DBN), an unsupervised greedy layer-by-layer training algorithm is proposed to bring about the hope of solving the deep structure-related optimization problems. Then the deep structure of multi-layer automatic encoder is proposed. Besides, the convolutional neural network proposed by Lecun et al. is the first right multi-layer structure learning algorithm [5], which uses relative spatial relations to reduce the number of parameters to improve training performance. In deep learning, a neural network is a bio-incentive model
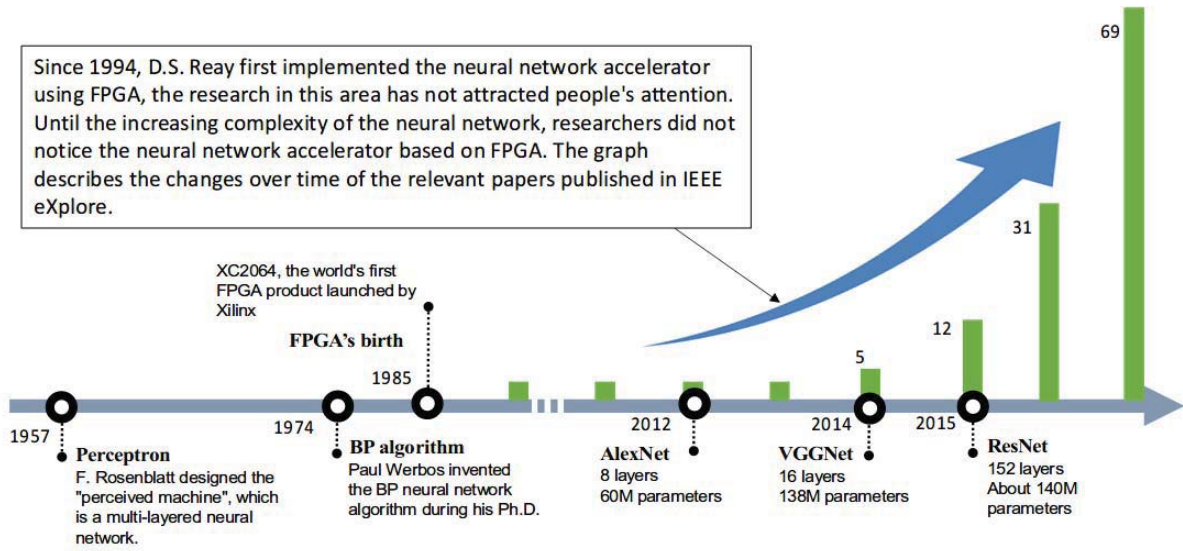
Fig. 1: Development history of the neural network accelerator based on FPGA.

that typically includes multiple layers of neurons, and different algorithms are combinations between different network layers. Each layer receives the neurons of the previous layer as input.

III. STATE-OF-THE-ART DEVELOPMENTS

A. Acceleration methods

At present, the acceleration methods for neural networks are mainly divided into two types, software design optimization, and hardware design improvement. The primary goal of software design optimization is to reduce the computation or bandwidth requirements of the neural network model with keeping accuracy. In general, there are roughly three ways, which are optimization of algorithm procedure[6], data quantification and weight reduction[7]. The optimization of algorithm procedure is mainly for the characteristics of different neural network models, and the calculation process is simplified or transformed without affecting the result, thereby achieving the purpose of reducing the computation and reducing the bandwidth requirement. Data quantification is primarily the quantification of weights and neurons to reduce the bandwidth and storage requirements in neural network computing. Moreover, the last one, weight reduction, is to use a low-rank matrix to approximate the weight matrix so that the actual weight is reduced, reducing the total calculation of the model. The improvement of hardware design mainly points to the characteristics of neural network algorithms to improve the existing logical unit structure so that it can execute deep learning algorithms efficiently and quickly.[8, 9]

B. Acceleration platforms

At present, hardware platforms that can be accelerated for neural networks mainly include GPU, FPGA, and ASIC.

The primary survey here is the FPGA platform in this article. Although FPGA does not have the advantages of faster operation, lower power consumption, and cheaper mass production in ASIC platforms under the same design, due to their editable logic arrays, FPGA have shorter design cycles. For the GPU, because of the CUDA general-purpose parallel computing framework, the design solution is also very convenient and fast, but the power consumption of the GPU is higher. So comprehensively, for the same parallel implementation, using the FPGA platforms can achieve good performance and efficiency improvement in a shorter time.

C. The focus of the neural network algorithm acceleration

According to the results of previous surveys, most of the research relies on the roofline model[16] for the theoretical analysis of FPGA accelerators. In this model, the X-axis of the roofline model means the system's computational communication ratio, CTC, and the Y-axis is the peak computing power of the system. This model describes the relationship between the computing power of the system and the communication bandwidth. For the previous research, they mainly realized the increase of CTC ratio from the following aspects, to achieve the purpose of optimizing the neural network accelerator.

1) Common characteristics of the algorithm: For many neural network algorithms, the impact of different parts of the algorithm on the execution time of the entire algorithm is different during execution. Nonetheless, for any neural network algorithm, there are many common features, and a more general accelerator can be designed for these features. The common features of each neural network algorithm are matrix operations, nonlinear activation functions and huge internal parameters. For the matrix calculation in the neural
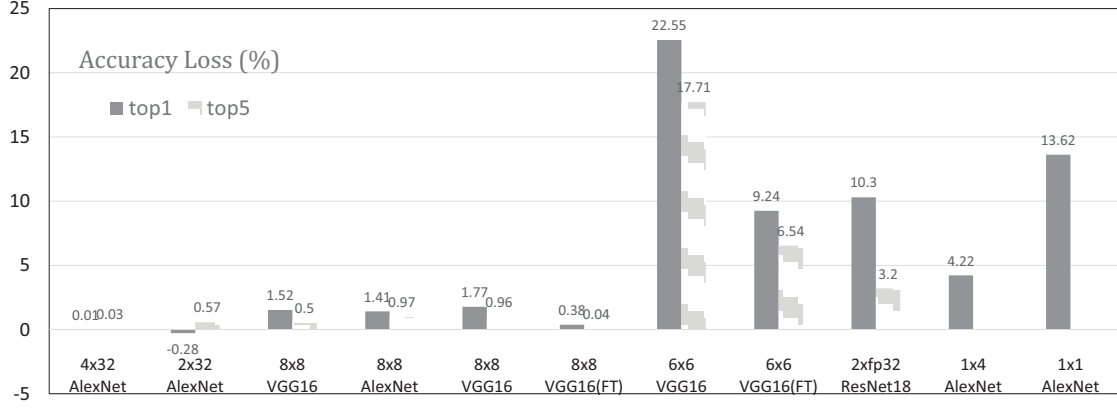
Fig. 2: Comparison of different data quantification methods[7, 10–15]. The experiments in label mean $BW_{weight} \times BW_{neurons}$ and FT indicates that the network is adjusted after quantized.

network, there are im2col, Winograd-based method[9], loop unrolling and matrix sparsity analysis, which can increase the data multiplexing of each calculation. Thereby it can reduce the total number of memory accesses and increase the CTC ratio[6, 17].

In figure 2, the meaning of the abscissa is each network with information which is (number of bits of the weight matrix) x (number of bits of the neuron matrix), and FT indicates that the network is adjusted after quantified. The ordinate is the loss of the correct rate of each model after the changes to the original model. It can be seen that for linear quantization if we want to achieve the accuracy without losing its loss after the transformation, it is best not to reduce the bit of data to less than eight digits.

2) Parallel Neural Network Algorithm: Computational parallelization is the most commonly used acceleration method for neural network algorithms. The use of task-level parallelism, data-level parallelism, and hardware-level parallelism is the main parallel processing of accelerator optimization. In essence, the essence of the parallel neural network algorithm is to parallelize the calculation of the core of the algorithm, in order to achieve better acceleration[18]. Task-level parallelism involves the optimization of software systems[19]. Due to the various open frameworks used in various studies, no relevant resources have been found for the time being. For data-level parallelism, many studies now use a double-buffer mechanism to speed up the entire computation time by cover the time cost of data transmission at the time of computation in the computational unit[6].

There are many ideas in hardware-level parallelism, such as calculation units for reconfiguring each layer, and a com-

promise parameter scheme for all layers[9], or a hardware platform with different parameter combinations configured in advance[20]. There is a solution that accelerates by thermally switching to different platform calculations in the computational process. Also, pipeline technology is often used in accelerators to increase throughput[20].

IV. DESIGN OF FPGA-BASED DEEP LEARNING ACCELERATORS

This section is mainly a summary of the previous research on the papers of the neural network accelerator based on FPGA. In general, the starting point of the problems and the accelerator schemes designed by these papers are very different. However, depending on the type of problems to be solved, the current researches are classified into four categories, which are specific to each: accelerators for a specific application, accelerators for specific algorithms, accelerators for common features of algorithms, and general accelerator frameworks with hardware templates.

A. Designing Accelerators for Specific Applications

Utilizing FPGA design accelerators for specific problems is currently the most extensive area of FPGA accelerator applications. Designing an accelerator specifically for a specific problem, it not only fits the problem well but also has a relatively small design difficulty. Designing accelerators for specific problems often speed up the reasoning process of deep learning algorithms rather than the learning process.

The paper [21] used FPGA to design a dedicated acceleration device to implement the LSTM algorithm to achieve an efficient speech recognition engine (ESE). To speed up predictions and save energy, they used a load-balanced sensing

pruning method that compresses the LSTM model size by 20x (10x form pruning, 2x form quantization) with negligible loss of prediction accuracy. The compressed model is then encoded and split into multiple PEs for parallelism, and a complex LSTM data stream is scheduled using a separately designed scheduler. Finally, an ESE hardware architecture that directly runs the sparse LSTM model is implemented. The ESE is implemented in a Xilinx XCKU060 FPGA operating at 200MHz and operates directly on a sparse LSTM network with a performance of 282GOPS, corresponding to a 2.52 TOPS on a dense LSTM network. Moreover, it processes a full LSTM for speech recognition with a power dissipation of 41 Watts. Evaluated on the speech recognition benchmark LSTM, ESE is 43x faster and 3x faster than the Core i7 5930k CPU and Pascal TitanX GPU. Compared with the CPU and GPU, the energy efficiency of 40x and 11.5x is improved respectively.

B. Designing Accelerators for Specific Algorithms

The use of FPGA design accelerators for a specific neural network model algorithm is currently a hot research topic in the direction of accelerators. The main reason is that when an accelerator designed for a specific neural network algorithm is applied to a specific problem, it usually only needs to be configured with specific parameters or some small changes to fit this problem well.

1) Convolutional neural network: The paper[22] argues that the accelerator still has a significant problem, that is, the computational process may not match the FPGA platform memory bandwidth very well. As a result, existing methods fail to achieve optimal performance due to failure to take full advantage of logical resources or memory bandwidth. At the same time, the increasing complexity and magnitude of deep learning applications exacerbate this problem. To overcome this problem, they use the analytical design mechanism of the roofline model. For any CNN design, quantitatively analyze its computational memory and required memory bandwidth by using different optimization methods, such as loop un-rolling and transformation. Moreover, then, using the roofline model approach, the solution with the best performance and lowest FPGA resource requirements was confirmed. By the paper[6], the double-buffer mechanism is used to optimize the memory access, and the CTC is further improved. Finally, a convolutional neural network accelerator is implemented on the VC707 FPGA platform. The final result of the experiment was that the peak performance of the hardware reached 61.62 GFLOPS at 100 MHz operating frequency. The double-buffer mechanism and the unique computing unit of the paper can be used in other algorithms to further realize the vision of the universal accelerator for neural networks.

2) Recurrent neural network: Long short-term memory recurrent neural networks (LSTM-RNs) is a kind of Recurrent neural networks (RNNs). In this paper[23], they proposed an FPGA-based LSTM-RNs accelerator that optimizes comput-ing performance and communication requirements. The peak performance of the accelerator is finally achieved at 7.26 GFLOP/S. It can be seen that the accelerator structure of the

paper[23] is the same as that of the paper[6], except that the LSTM buffer area is added in the middle to directly save the state parameters of the LSTM hidden layer, eliminating the need to reload the parameters each time. , reducing the bandwidth requirements for hardware. The idea can be generalized by adding a direct cache between each layer so that the next level of computation does not require data loading in the process of computing a neural network model.

C. Design Accelerator for Common Features of Algorithms

The purpose of the two previous design accelerators is dedicated, but there are some practices in the process of designing accelerators that can be ported to accelerator designs for other algorithms. Thus, the accelerator designed for the characteristic features of the algorithm can be better imple-mented. At present, according to the papers that are known, their general methods for processing accelerator design are computational optimization and memory access optimization.

1) Calculation Optimization: Most of the deep learn-ing algorithms involve a large number of large-scale matrix operations in the process of learning or reasoning. These matrix operations generally require a large number of com-puting resources, so they are often the core of the algorithm. Therefore, accelerating the matrix operations involved can effectively improve the overall performance of the algorithm. The paper[6] optimizes the loop code of the matrix operation. First, the loop unrolling method is used to convert the original loop code, and according to the relationship between the loop variables of each loop and the arrays in the loop, the following three categories are divided:

1) Irrelevant. If a loop variable i_k does not appear in any access function of array A, then the corresponding loop dimension is said to be irrelevant to array A.
2) Independent. If the data space set is accessed on array A is completely separable by a certain loop size i_k or for any given two distinct parameters p_1 and p_2. For example, the access data $DS(A, i_k = p_1) = \bigcup Image(F_S^A, (D_S \cap i_k = p_1))$ is disjoint with $DS(A, i_k = p_2) = \bigcup Image(F_S^A, (D_S \cap i_k = p_2))$, then the loop size ik Is independent of array A.
3) Dependent. If the data space set on array A is accessed non-divisible by a certain loop size i_k, then the loop size i_k is dependent with array A.

The paper used this method to select parameter and design hardware to achieve the effect of hardware acceleration. However, for the optimization of the calculation part, primarily every research has involved, and it can be roughly classified into the above method. Another is to use the sparsity of the weight matrix, and it performs the calculation skip operation when detecting the value of weight is 0 [17].

2) Memory access optimization: Design variables with higher computational roof do not necessarily achieve higher performance under the constraints of memory bandwidth. The paper[17] has also been optimized to reduce traffic with efficient data multiplexing. Code 1 illustrates the memory transfer operation of the CNN layer. The feature map and

weight of the input and output are loaded before the calculation engine starts, and the output feature map is written back to the main storage. If the most inner loop of the communication part (the loop size t_i in Code 1 is independent of the array, there will be redundant memory operations between different loops. The most inner loop variable t_i is independent of the array output_fm. Thus, for accessing the array output_fm can be placed into the outer loop. With this memory optimization, the total count of array output_fm memory access operations is reduced from $\frac{2*M*N*R*C}{T_m*T_n*T_r*T_c}$ to $\frac{M*R*C}{T_m*T_r*T_c}$

CODE 1 Computing Flow of Convolution Layer

INPUT: The feature map of last layer, input_fm, (S*R+K, S*C+K, N); The convolution kernel, weights, (K, K, M); The size of stride, S

OUTPUT: The feature map of this layer, output_fm, (R, C, M)

```
 1: function Cal_CONV
 2:     for row = 0 to R by T_r do
 3:         for col = 0 to C by T_c do
 4:             for tm = 0 to M by T_m do
 5:                 for tn = 0 to N by T_n do
 6:                     //load output feature maps
 7:                     //load weights
 8:                     //load input feature maps
        L: foo(output_fm(tm, row, col), weights(tm, tn), input_fm(tn, row, col));
 9:                     //store output feature maps
10:                 end for
11:                 //store output feature maps
12:             end for
13:         end for
14:     end for
15: end function
```

D. Designing a Universal Accelerator Framework with Hardware Templates

Using a hardware template design accelerator is a more general approach than previous accelerator design methods. Generally, these hardware templates are often the implementation of the FPGA version of an individual programming model. In the process of using, the user only needs to design a small part of the module and configure the parameters. When the parameters and the module are determined, the accelerator framework can run automatically to speed up problems that users have to solve.

The paper [24]proposed FP-DNN (Field Programmable DNN), an end-to-end framework, which uses the DNN described by TensorFlow as input, and automatically generates hardware implementation on the FPGA board with RTL-HLS hybrid template. The implementation of the paper in Table 1 is compared to the implementation on the CPU and GPU. At the same time, multiple neural network models were selected as benchmarks: VGG[1], LSTM[35], and Res-Net[36]. They used both data quantization strategies in these three models and compared the model accuracy of 32-bit floating point and 16-bit fixed points in Table 1. The top-5 precision of VGG-19 and Res-152 was tested on the ImageNet data-set for evaluation. The LSTM-LM model was evaluated using perplexity of the model tested on the PTSB data set. The lower perplexity means the better model performance in the language modeling task. It can be seen that, while maintaining the same calculation result, the 16-bit fixed point computing performance of the FPGA platform is basically about $2\times$-$3\times$ of the CPU platform, and the energy utilization rate is also nearly $20\times$. It thoroughly explains the rationality of the framework of the paper.

For the research of the general hardware accelerator framework, there are still few studies and no more information. There is a need to further validate the methods of research in this area.

E. Comprehensive comparison of current accelerator performance

In Table 1, we could observe the performance and power consumption of the current mainstream FPGA-based neural network accelerators under different network models, different hardware types and different external parameters. Firstly, according to the types of network models used, this paper divides them into three categories: VGG, LSTM, and Res-Net. Moreover, the precision of the parameters of the model used are given. At the same time, the table also lists the platform, hardware types and related parameters in these paper. At the end of the table, the experimental results are shown, which are GOP/s, GOP/j and power respectively. It can be seen from the table that whether increasing frequency, changing memory types and reducing parameter precision appropriately have a positive impact on the accelerator. Among these papers, the impressive one is multi-FPGA cluster used in the paper[31], which contains 15 pieces of FPGA chips. By effectively connecting 15 FPGAs with the workload and weight balancing, an average of about 1200 GOP/s and 38GOP/j per chip of FPGA is achieved. It is even better than that of the neural network accelerator with single FPGA in the GOP/s, and also achieves high energy utilization. It provides a new way of thinking for the current research.

V. Oppotunies and Challenges

As early as the 1960s, Gerald Estrin proposed the concept of reconfigurable computing. It was not until 1985 that the first FPGA chip was introduced by Xilinx. With the continuous development of deep learning, due to the high parallelism of its applications, more and more researchers are investing in the research of FPGA-based deep learning accelerators, which is also the trend of the times.

A. Advantages of FPGA based accelerators

1) High performance with low energy: The advantage of high energy efficiency is not to be overstated, and many previous studies have shown this fact. It can also be seen from Table 1 that the GOP/j on the FPGA platform can reach tens of times on the CPU platform, and the lowest level is the same level of energy efficiency on the GPU platform.

TABLE I: Performance comparison of different networks on different platforms

Model	Platform	Specification			Precision	GOP/s	GOP/j	Power
		Types	Frequency	Memory				
VGG								
VGG-19[24]	CPU	Xeon E5-2650v2	2.6GHz	-	float32	119	0.63	95W
VGG-19[24]	GPU	GTX TITAN X	1002MHz	12G GDDR5	float32	1704	6.82	250W
VGG-16[15]	FPGA	Stratix-V GSD8	120MHz	32G DDR3	fixed8	117.8	6.17	19.1W
VGG-16[2]	FPGA	Stratix-V GSD8	200MHz	on-chip	fixed16	821	-	-
VGG-16[25]	FPGA	Arria 10 SX660	120MHz	- DDR4	8-bit	53	13.9	3.3W
VGG-16[26]	FPGA	Arria 10 GX 1150	150MHz	8G DDR3L	fixed8/16	645.25	-	-
VGG-16[27]	FPGA	Arria 10 GX 1150	240MHz	- DDR3	fixed8/16	968.03	-	-
VGG-16[27]	FPGA	Stratix 10 GX 2800	300MHz	- DDR3	fixed8/16	1604.57	-	-
VGG[28]	FPGA	Arria 10 GX 1150	370MHz	1G DDR4	float-	866	20.75	19.1W
VGG[28]	FPGA	Arria 10 GX 1150	385MHz	1G DDR4	fixed16	1790	47.78	-
VGG-S[29]	FPGA	XCKU115	125MHz	off-chip	fixed32	394.7	14.6	27W
VGG-D[30]	FPGA	Virtex 7 VX690T	200MHz	off-chip	fixed8	1467.6	-	-
VGG-A[30]					fixed8	1500	-	
VGG-16[31]	15xFPGAs	XC7VX690T	-	off-chip	fixed16	1197*	37.88	-
VGG-19[31]					fixed16	1220*	38.13	
VGG-19[24]	FP-DNN	Stratix-V GSMD5	150MHz	4G DDR3	float32	81	3.24	25W
					float16	364.36	14.57	
LSTM								
LSTM-LM[24]	CPU	Xeon E5-2650v2	2.6GHz	-	float32	103	0.54	95W
LSTM-LM[24]	GPU	GTX TITAN X	1002MHz	12G GDDR5	float32	1828	7.31	250W
LSTM[21]	FPGA	XCKU060	200MHz	8G DDR3	fixed16/12	282.2	6.87	41W
LSTM[23]	FPGA	Virtex7-485t	150MHz	- DDR3	float32	7.26	-	19.63W
Bi-LSTM[32]	FPGA	Zynq XCZU7EV	266MHz	on-chip	fixed1/8	1833	-	-
LSTM-LM[24]	FP-DNN	Stratix-V GSMD5	150MHz	4G DDR3	float32	86	3.44	25W
					float16	315.85	12.63	
Res-Net								
Res-152[24]	CPU	Xeon E5-2650v2	2.6GHz	-	float32	119	0.63	95W
Res-152[24]	GPU	GTX TITAN X	1002MHz	12G GDDR5	float32	1661	6.60	250W
Res-152[33]	FPGA	Arria 10 GX 1150	150MHz	-	float16	315.5	-	-
Res-50[33]	FPGA	Arria 10 GX 1150	150MHz	-	float16	285.07	-	-
Res-50[2]	FPGA	Stratix-V GSD8	200MHz	on-chip	fixed16	973	-	-
Res-50[34]	FPGA	Stratix^{TM} 10	750MHz	-	float32	15000	85	-
Res-50[27]	FPGA	Arria 10 GX 1150	240MHz	- DDR3	fixed8/16	599.61	-	-
Res-152[27]					fixed8/16	697.09	-	
Res-50[27]	FPGA	Stratix 10 GX 2800	300MHz	- DDR3	fixed8/16	651.49	-	-
Res-152[27]					fixed8/16	789.44	-	
Res-152[24]	FP-DNN	Stratix-V GSMD5	150MHz	4G DDR3	float32	73	2.92	25W
					float16	226.47	9.06	

* represents that the value is the measured value of each FPGA

2) High parallelism: High parallelization is the main property of choosing an FPGA platform to accelerate deep learning. Thanks to the FPGA's editable logic hardware unit, we can easily optimize the hardware with the parallelization algorithm to achieve high parallelism.

3) Flexibility: Due to the reconfigurability of the FPGA, it could be applied to complex engineering situations. For instance, after the hardware design and application design is completed, it is found through experiments that the performance does not reach the ideal situation. Reconfigurability enables FPGA-based hardware accelerators to handle frequent design changes well and satisfy the changing needs of users. Therefore, this flexibility is also a bright spot on FPGA platforms compared to ASIC platforms.

B. Disadvantages of FPGA based accelerators

1) Reconfigurable Cost: The reconfigurability of the FPGA platform is also a double-edged sword. Although it gives us many advantages in computational acceleration, it cannot be ignored that the cost of time in reconfiguration. Generally, the reconfiguration process is divided into two types: static reconfiguration and dynamic reconfiguration. Static reconfiguration, also known as compile-time reconfiguration, refers to the ability to configure the hardware to configure one or more functions of the system before the task runs, and locks it before the task finishes. The other is also known as runtime reconfiguration. Dynamic reconfiguration of hardware using context configuration mode. During the execution of the task, the hardware module is reconfigured as needed, but it is very susceptible to delays, and increase the runtime.

2) Programming Difficulty: Although the concept of reconfigurable computing architecture has long been proposed, and there has been much more mature work, reconfigurable computing has not gained popularity before. The reason is that with a mature system, traditional programming on the CPU adopts high-level abstract programming languages. However, reconfigurable computing requires hardware programming, generally using hardware programming Languages (Verilog, VHDL.) that would cost programmers much time to master.

C. Expectation

Although the neural network accelerator based on FPGA still has such and such problems, the future development is expected. Through the overview of this article, the following issues need further study in this direction:

1) Optimization in the rest of the computation process. At present, mainstream research is placed in the loop part of the matrix operation, and the calculation of the activation function is only a few people involved.
2) Access optimization. Further research is needed for other optimization methods for data access.
3) Data optimization. Using lower bit data can naturally improve the performance of the platform, but most of them make weights and neurons in same bit width, but the difference bit width with the non-linear map can also

be improved in figure 2. So there should be a better balance status to be explored.
4) The integration of FPGAs. According to the performance of the paper[31], if the problems of scheduling and allocation can be handled well, the multi-FPGA cluster can achieve better results. Moreover there is not much research on this direction at present. So it is worth for this direction to explore further.
5) Automatic configuration. To solve the problem of complex programming on the FPGA platform, if there is a more user-friendly automatic deployment framework, similar to NVIDIA's CUDA (Compute Unified Device Architecture), it will make the application scope wider.

VI. Conclusion

Accelerating deep learning algorithms is a study that has increased many attentions in recent years. The current mainstream platform is the GPU cluster. Although FPGA/ASIC also has such a good acceleration capability, it is only popularized in the research field due to programming complexity and other issues. In this survey, we have investigated the design and implementation of the FPGA based accelerators by the order from customized to general, compared the performance and power consumption of different designs, and summarized some directions for further research in this field.

VII. Acknowledgment

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.
[2] R. Zhao, H.-C. Ng, W. Luk, and X. Niu, "Towards efficient convolutional neural network for domain-specific applications on fpga," in 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018, pp. 147–1477.
[3] C. Wang, X. Li, Y. Chen, Y. Zhang, O. Diessel, and X. Zhou, "Service-oriented architecture on fpga-based mpsoc," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 10, pp. 2993–3006, 2017.
[4] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, vol. 18, no. 7, pp. 1527–1554, 2006.
[5] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in Proceedings of 2010 IEEE International Symposium on Circuits and Systems. IEEE, 2010, pp. 253–256.
[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional

neural networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2015, pp. 161–170.

[7] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song et al., "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, 2016, pp. 26–35.

[8] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 3, pp. 513–517, 2017.

[9] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019.

[10] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 35–47, 2018.

[11] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[12] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," arXiv preprint arXiv:1605.04711, 2016.

[13] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefanet: Training low bitwidth convolutional neural networks with low bitwidth gradients," arXiv preprint arXiv:1606.06160, 2016.

[14] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," arXiv preprint arXiv:1612.01064, 2016.

[15] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016, pp. 16–25.

[16] Y. Lo, S. Williams, B. Straalen, T. Ligocki, M. Cordery, N. Wright, M. Hall, and L. Oliker, "Roofline: an insightful visual performance model for multicore architectures," High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation, vol. 8966, pp. 129–148, 2015.

[17] L. Qinrang and L. Chongyang, "Calculation optimization for convolutional neural networks and fpga-based accelerator design using the parameters sparsity," Journal of Electronics & Information Technology, vol. 40, no. 6, pp. 1368–1374, 2018.

[18] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Maloc: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2601–2612, 2018.

[19] C. Wang, J. Zhang, X. Li, A. Wang, and X. Zhou, "Hardware implementation on fpga for task-level parallel dataflow execution engine," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 8, pp. 2303–2315, 2016.

[20] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in Proceedings of the 2016 International Symposium on Low Power Electronics and Design. ACM, 2016, pp. 326–331.

[21] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang et al., "Ese: Efficient speech recognition engine with sparse lstm on fpga," in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017, pp. 75–84.

[22] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional

neural networks on fpgas," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC). IEEE, 2017, pp. 1–6.

[23] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2017, pp. 629–634.

[24] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017, pp. 152–159.

[25] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "Fpga-based cnn inference accelerator synthesized from multi-threaded c software," in 2017 30th IEEE International System-on-Chip Conference (SOCC). IEEE, 2017, pp. 268–273.

[26] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017, pp. 45–54.

[27] ——, "Automatic compilation of diverse cnns onto high-performance fpga accelerators," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[28] J. Zhang and J. Li, "Improving the performance of opencl-based fpga accelerator for convolutional neural network," in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017, pp. 25–34.

[29] S. Huang, J. Jiang, Y. Dou, L. Bai, H. Wang, and B. Qin, "Design and implementation of convolutional neural network accelerator with variable layer-by-layer debugging," in Proceedings of the 2018 2nd International Conference on Deep Learning Technologies. ACM, 2018, pp. 1–6.

[30] J. Yu, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang, "Instruction driven cross-layer cnn accelerator with winograd transformation on fpga," in 2017 International Conference on Field Programmable Technology (ICFPT). IEEE, 2017, pp. 227–230.

[31] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, "A framework for acceleration of cnn training on deeply-pipelined fpga clusters with work and weight load balancing," in 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018, pp. 394–3944.

[32] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "Finn-l: Library extensions and design trade-off analysis for variable precision lstm networks on fpgas," in 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018, pp. 89–897.

[33] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J.-s. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in 2017 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2017, pp. 1–4.

[34] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra et al., "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017, pp. 5–14.

[35] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," arXiv preprint arXiv:1409.2329, 2014.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.