Optimal Design of Neural Networks Based on FPGA

Subjects: Computer Science, Hardware & Architecture
Contributor: Chenghao Wang, Zhongqiang Luo

Deep learning based on neural network has been widely used in image recognition, speech recognition, natural language processing, automatic driving and other fields, and has made breakthrough progress. FPGA stands out in the field of accelerated deep learning with its flexible architecture and logical unit, high energy efficiency ratio, strong compatibility, low delay and other advantages.

Keywords: deep learning ; deep neural network ; FPGA

1. Introduction

With the rise of the short video industry and the advent of the era of big data and the Internet of Things (IoTs), the data created by people in recent years has shown a blowout growth, providing a solid data foundation for the development of artificial intelligence (AI). As the core technology and research direction for realizing artificial intelligence, deep learning based on neural networks has achieved good results in many fields, such as speech recognition [1][2][3], image processing [4][5][6], and natural language processing [7][8][9]. Common platforms to accelerate deep learning include central processing unit (CPU), graphics processing unit (GPU), field-programmable gate array (FPGA), and application-specific integrated circuit (ASIC).

Among them, the CPU adopts Von Neumann architecture and the program execution in the field of deep learning of artificial intelligence is less, while the computational demand for data is relatively large. Therefore, the implementation of AI algorithms by CPU has a natural structural limitation—that is, the CPU spends a large amount of time reading and analyzing data or instructions. In general, it is not possible to achieve unlimited improvement of instruction execution speed by increasing CPU frequency and memory bandwidth without limit.

The ASIC special purpose chip has the advantages of high throughput, low latency, and low power consumption. Compared with FPGA implementation of the same process, ASIC can achieve 5~10 times the computation acceleration, and the cost of ASIC will be greatly reduced after mass production. But, deep learning computing tasks are flexible, each algorithm's needs can be implemented effectively through slightly different dedicated hardware architecture, and there is a high cost of research and development of ASIC and flow, with the cycle being long, yet the most important thing to note is that is logic cannot cause dynamic adjustment, which means that the custom accelerator chips (such as ASIC) must make a large amount of compromise in order to act as a sort of universal accelerator.

At a high level, this means that system designers face two choices. One is to select a heterogeneous system and load a large number of ASIC accelerator chips into the system to deal with various types of problems. Or, one can choose a single chip to handle as many types of algorithms as possible. The FPGA scheme falls into the latter category because FPGA provides unlimited reconfigurable logic, whereas FPGA can update the logic function in only a few hundred milliseconds.

At present, artificial intelligence computing requirements represented by deep learning mainly use GPU, FPGA, and other existing general chips that are suitable for parallel computing in order to achieve acceleration. Due to the characteristics of high parallelism, high frequency, and high bandwidth, GPU can parallelize the operation and greatly shorten the operation time of the model. Due to its powerful computing ability, it is mainly used to deal with large-scale computing tasks at present. GPU-accelerated deep learning algorithms have been widely used and have achieved remarkable results.

Compared with FPGA, the peak performance of GPU (10 TFLOPS, floating point operation per second) is much higher than that of FPGA (<1 TFLOPS), and thus GPU is a good choice for deep learning algorithms in terms of accelerating performance. However, this is only the case when the power consumption is not considered. Because the power consumption of GPU is very high, sometimes tens of hundreds of times that of CPU and FPGA, the high energy

consumption limits it to being used in high-performance computing clusters. If it is used on edge devices, the performance of GPU will be greatly sacrificed.

However, the DDR (double data rate) bandwidth, frequency, and number of computing units (low-end chips) of FPGA are not as high as GPU. For on-chip memory, FPGA has larger computing capacity; moreover, on-chip memory is crucial for reducing latency in applications such as deep learning. Accessing external memory such as DDR consumes more energy than the chip itself for computing. Therefore, the larger capacity of on-chip cache reduces the memory bottleneck caused by external memory reading and also reduces the power consumption and cost required by high-memory bandwidth. The large capacity of on-chip memory and flexible configuration capability of FPGA reduces the read and write of external DDR, while GPU requires the support of an external processor during operation. The addition of external hardware resources greatly reduces the data processing speed. Moreover, FPGA's powerful raw data computing power and reconfigurability allow it to process arbitrary precision data, but GPU's data processing is limited by the development platform. Therefore, in this context, FPGA seems to be a very ideal choice. In addition, compared with GPU, FPGA not only has the characteristics of data parallel, but also has the characteristics of pipeline parallel, and thus for pipelined computing tasks, FPGA has a natural advantage in delay. On the basis of these advantages, FPGA stands out in the field of accelerated deep learning.

In the previous work, the application, acceleration method and accelerator design of neural networks such as CNN (convolutional neural network), RNN (recurrent neural network), and GAN (generative adversarial network) based on FPGA were described in detail, and the research hotspots of industrial application combined with FPGA and deep neural networks were fully investigated. They have made relevant contributions to the application and development of FPGA in neural networks [10][11][12][13][14]. It is worth noting that the analysis of the acceleration effect of different acceleration techniques on different neural networks is less involved in their work.

2. The Development of Neural Networks

As shown in **Table 1**, the development process of neural networks can be roughly divided into five stages: the proposal of the model, the stagnation period, the rise of the back propagation algorithm, the confusion period, and the rise of deep learning.

Table 1. Development history of neural networks.

Stage	Year	Character	Content
Generation of models	1943	Warren McCulloch and Walter Pitts	McCulloch–Pitts model [15]
	1948	Alan Mathison Turing	B-type Turing machine [16]
	1949	Donald Hebb	Hebb algorithm [17]
	1951	McCulloch and Marvin Minsky	The first neural network machine SNARC
	1958	Frank Rosenblatt	Perceptron model [18]
Lag phase	1969	Marvin Minsky	Perception [19]
	1974	Paul J. Werbos	Backpropogation (BP) algorithm [20]
	1980	Kunihiko Fukushima	Neocognitron model [21]
	1982	John J. Hopfield	Hopfield model [22]
The rise of backpropagation algorithms	1985	Hinton and Sejnowski	Boltzmann machine [23]
	1986	David Rumelhart and James McClelland	Redescription of the BP algorithm [24]
	1989	LeCun	Introducing the BP algorithm to the convolutional neural network [25]
Confusion period	1990–2005	The rise of machine learning models has brought about great challenges to the development of neural networks	

Stage	Year	Character	Content
The rise of deep learning	2006	Geoffrey Hinton	Deep Belief Networks [26]
	2012	Alex Krizhevsky	AlexNet [27]
		Christian Szegedy	GoogLeNet [28]
		Visual Geometry Group and Google DeepMind	VGGNet [29]
	2014	Ian J. Goodfellow	GAN [30]
		Yi Sun and Xiaogang Wang	DeepID [31]
		Ross Girshick and Jeff Donahue	Region-CNN (RCNN) [32]
	2015	Joseph Redmon	You Only Look Once (YOLOv1) [33]
	2016	AlphaGo, an artificial intelligence machine developed by Google's DeepMind, beat Go world champion Lee Sedol 4-1	
		Joseph Redmon	YOLOv2 [34]
	2018	Joseph Redmon	YOLOv3 [35]
	2020	Alexey Bochkovskiy	YOLOv4 [36]
	2020–2022	YOLOv5, YOLOv6 [37], YOLOv7 [38], etc.	

2.1. Deep Neural Network (DNN)

Since a single-layer perceptron cannot solve linear inseparability problems, it cannot be used in industry. Scholars have expanded and improved the perceptron model by increasing the number of hidden layers and corresponding nodes to enhance the expression ability of the model, and thus the deep neural network (DNN) was created. The DNN is sometimes called a multi-layer perceptron (MLP). The emergence of the DNN overcomes the low performance of a single-layer perceptron. According to the position of different layers in the DNN, the neural network layers inside the DNN can be divided into three types: the input layer, hidden layer, and output layer, as shown in **Figure 2**.

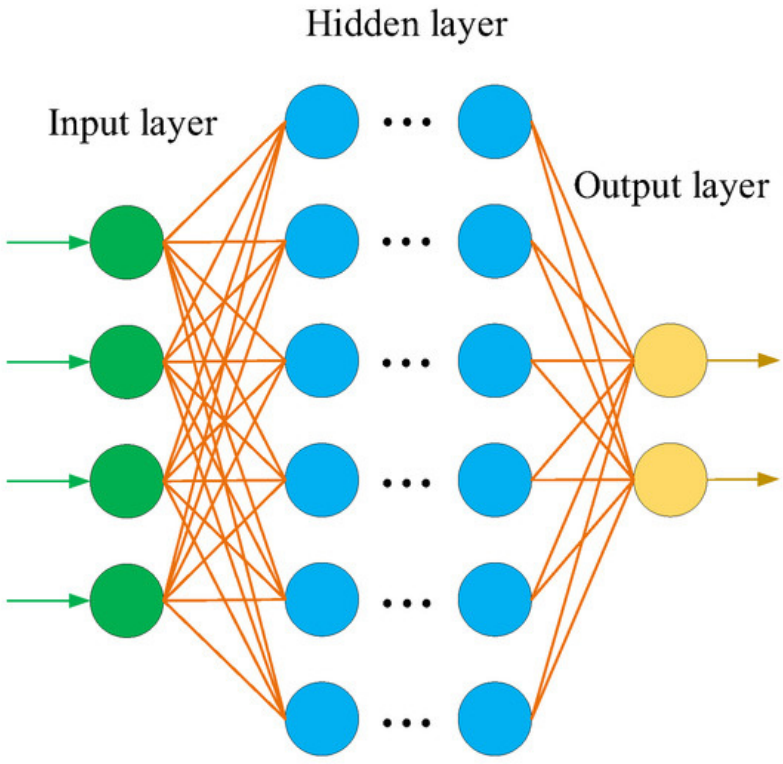

Figure 2. Typical model of a DNN.

2.2. Convolutional Neural Network (CNN)

The CNN model was developed from the early artificial neural network. On the basis of the research of Hub et al. on the cells in the visual cortex of cats, the CNN model is a specially designed artificial neural network with multiple hidden layers through the biomimetic brain skin layer. Convolution operation is used to solve the disadvantages of large computation and loss of structure information of artificial neural networks.

The birth of LeNet-5 established the basic embryonic form of CNN, which is composed of a convolution layer, a pooling layer, an activation function, and a fully connected layer connected in a certain number of sequential connections, as shown in **Figure 3**. The CNN is mainly applied to image classification [39][40][41], object detection [42][43], and semantic segmentation [44][45][46], as well as in other fields. The most common algorithms are YOLO and R-CNN, among which YOLO has a faster recognition speed due to the characteristics of the algorithm. It has been upgraded to V7. R-CNN's target location search and identification algorithm are slightly different from those of YOLO. Although the speed is slower than that of YOLO, the accuracy rate is higher than that of YOLO.

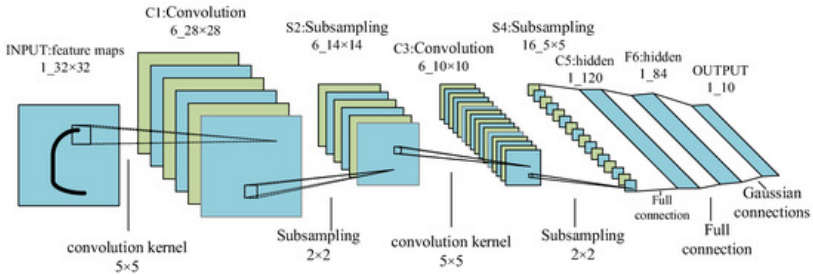

Figure 3. The Lenet-5 model.

2.3. Recurrent Neural Network (RNN)

Different from CNN, RNN introduces the dimension of "time", which is suitable for processing time-series-type data. Because the network itself has a memory ability, it can learn data types with correlation before and after. Recurrent neural networks have a strong model fitting ability for serialized data and are widely used in the field of natural language processing (NLP), including image classification, image acquisition, machine translation, video processing, sentiment analysis, and text similarity calculation. The specific structure is as follows: the recurrent neural network will store and remember the previous information in the hidden layer and then input it into the current calculation of the hidden layer unit.

Figure 4 shows the typical structure of an RNN, which is similar to but different from the traditional deep neural network (DNN). The similarity lies in that the network models of DNN and RNN are fully connected from the input layer to the hidden layer and then to the output layer, and the network propagation is also sequential. The difference is that the internal nodes of the hidden layer of the RNN are no longer independent of each other but have messages passing to each other. The input of the hidden layer can be composed of the output of the input layer and the output of the hidden layer at a previous time, which indicates that the nodes in the hidden layer are self-connected. It can also be composed of the output of the input layer, the output of the hidden layer at the previous moment, and the state of the previous hidden layer, which indicates that the nodes in the hidden layer are not only self-connected but also interconnected.

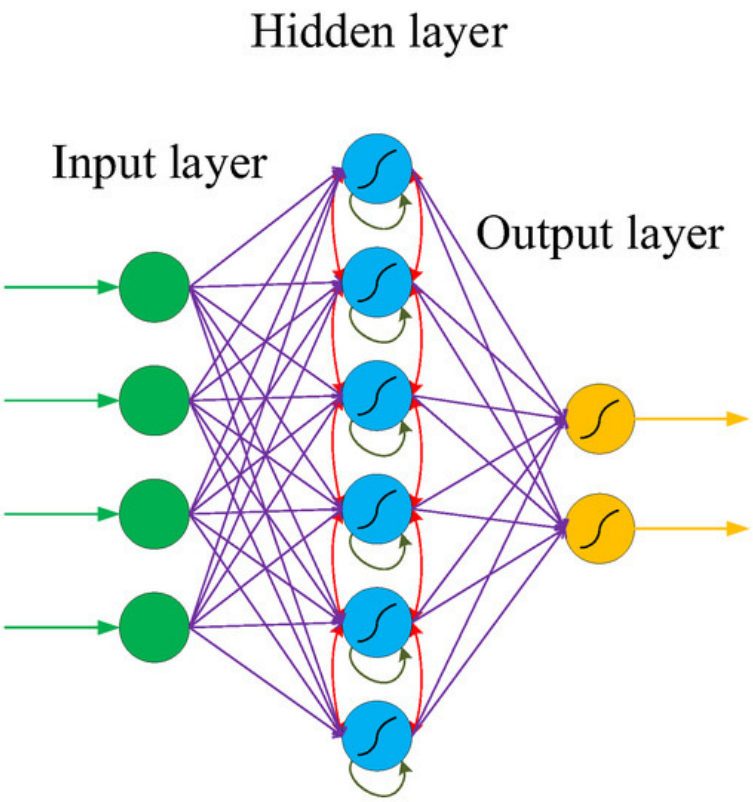

Figure 4. Typical structure of an RNN.

2.4. Generative Adversarial Network (GAN)

The GAN (generative adversarial network) is a machine learning model designed by Goodfellow et al. [30] in 2014. Inspired by the zero-sum game in game theory, this model views the generation problem as a competition between generators and discriminators. The generative adversarial network consists of a generator and a discriminator. The generator generates data by learning, and the discriminator determines whether the input data are real data or generated data. After several iterations, the ability of the generator and discriminator is constantly improved, and finally the generated data are infinitely close to the real data so that the discriminator cannot judge whether they are true or false. The GAN is shown in **Figure 5**.

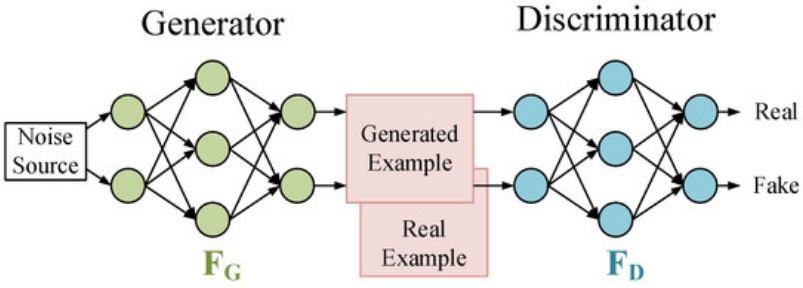

Figure 5. Typical structure of a GAN.

3. Application of Neural Networks Based on FPGA

3.1. Application of CNNs Based on FPGA

In terms of intelligent medical treatment, in November 2019, Serkan Sağlam et al. [47] deployed a CNN on FPGA to classify malaria disease cells and achieved 94.76% accuracy. In 2020, Qiang Zhang et al. [48] adopted a CPU + FPGA heterogeneous system to realize CNN classification of heart sound sample data, with an accuracy of 86%. The fastest time to classify 200 heart sound samples was 0.77 s, which was used in the machine of primary hospital to assist in the initial diagnosis of congenital heart disease.

In terms of national defense, in 2020, Cihang Wang et al. [49] found that a large number of high-precision and high-resolution remote sensing images were used in civil economic construction and military national defense, and thus they proposed a scheme of real-time processing of remote sensing images that was based on a convolutional neural network

on the FPGA platform. This scheme optimized CNN on FPGA from two aspects: spatial parallel and temporal parallel. By adding pipeline design and using the ideas of module reuse and data reuse, the pressure of the data cache was reduced, and the resource utilization of FPGA was increased. Compared with other schemes, the proposed scheme greatly improved the recognition speed of remote sensing images, reduced the power consumption, and achieved 97.8% recognition accuracy. Although the scheme uses many speedup techniques, it does not optimize the convolution operation, which has the most significant performance improvement.

3.2. Application of RNNs Based on FPGA

At present, in addition to the application of deep neural networks into the fields of image and video, an FPGA-based speech recognition system has also become a research hotspot. Among them, the commonly used speech recognition model is the RNN and its variant LSTM. Due to its huge market demand, speech recognition develops rapidly. In intelligent speech recognition products, in order to ensure certain flexibility and mobility, the speech recognition model is usually deployed on FPGA to meet the needs of intelligence and production landing.

In 2016, the work of J. C. Ferreira and J. Fonseca et al. [50] was considered as one of the earliest works to implement the LSTM network on the FPGA hardware platform, which focused people's attention from the software implementation of LSTM to FPGA hardware. In 2017, Y. Guan et al. [51] implemented the LSTM network using special hardware IP containing devices such as data scheduler, FPGA, AXI4Lite (one of the Advanced eXtensible Interfaces) bus, and optimized computational performance and communication requirements in order to accelerate inference.

In 2018, Zhe Li et al. [52] summarized two previous works on the FPGA implementation of the LSTM RNN inference stage on the basis of model compression. One work found that the network structure became irregular after weight pruning. Another involved adding a cyclic matrix to the RNN to represent the weight matrix in order to achieve model compression and acceleration while reducing the influence of irregular networks. On this basis, an efficient RNN framework for automatic speech recognition based on FPGA was proposed, called E-RNN. Compared with the previous work, E-RNN achieved a maximum energy efficiency improvement of 37.4 times, which was more than two times that of the latter work under the same accuracy.

In December of the same year, Chang Gao et al. [53] proposed a lightweight gated recursive unit (GRU)-based RNN accelerator, called EDGERDNN, which was optimized for low-latency edge RNN inference with batch size 1. EDGERDRNN used a delta network algorithm inspired by pulsed neural networks in order to exploit the temporal sparsity in RNN. Sparse updates reduced distributed RAM (DRAM) weight and memory access by a factor of 10, and reduced latency, resulting in an average effective throughput of 20.2 GOP/s for batch size 1.

3.3. Application of GANs Based on FPGA

In 2018, Amir Yazdanbakhsh et al. [54] found that generators and discriminators in GANs use convolution operators differently. The discriminator uses the normal convolution operator, while the generator uses the transposed convolution operator. They found that due to the algorithmic nature of transposed convolution and the inherent irregularity in its computation, the use of conventional convolution accelerators for GAN leads to inefficiency and underutilization of resources. To solve these problems, FlexiGAN was designed, an end-to-end solution that generates optimized synthesizable FPGA accelerators according to the advanced GAN specification. The architecture takes advantage of MIMD (multiple instructions stream multiple data stream) and SIMD (single instruction multiple data) execution models in order to avoid inefficient operations while significantly reducing on-chip memory usage. The experimental results showed that FlexiGAN produced an accelerator with an average performance of 2.2 times that of the optimized conventional accelerator. The accelerator delivered an average of 2.6 times better performance per watt compared to the Titan X GPU.

4. Neural Network Optimization Technology Based on FPGA

With the rapid development of artificial intelligence, the number of layers and nodes of neural network models is increasing, and the complexity of the models is also increasing. Deep learning and neural networks have put forward more stringent requirements on the computing ability of hardware. On the basis of the advantages of FPGA mentioned in the introduction, increasingly more scholars are choosing to use FPGA to complete the deployment of neural networks. As shown in **Figure 6**, according to different design concepts and requirements, FPGA-based neural network optimization technology can be roughly divided into optimization for data and operation, optimization for bandwidth, and optimization for memory and access, among others, which are introduced in detail below.

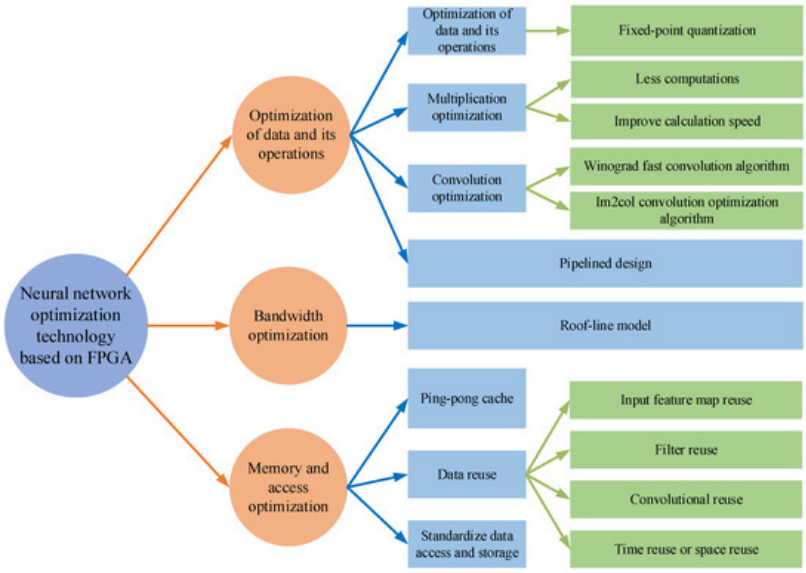

Figure 6. Neural network optimization technology based on FPGA.(fixed-point quantization [55][56][57][58][59][60][61][62], less computations [63][64][65], improve calculation speed [66][67][68][69], Winograd fast convolution algorithm [70][71][72][73][74][75], Im2col convolution optimization algorithm [76][77][78][79][80][81], pipelined design [82][83][84][85][86], Roof-line model [87][88][89], ping-pong cache [90][91][92][93], input feature map reuse [94][95], filter reuse [95][96], convolutional reuse [94][95][96], time reuse or space reuse [95], standardize data access and storage [97][98][99]).

4.1. Optimization of Data and Its Operations

In the aspect of optimization of data and its operation, scholars have made many attempts and achieved certain results. Aiming at the data itself, a method to reduce the data accuracy and computational complexity is usually used. For example, fixed-point quantization is used to reduce the computational complexity with an acceptable loss of data accuracy, so as to improve the computational speed. In terms of operation, the optimization is realized by reducing the computation times of multiplication and increasing the computation speed. The commonly used optimization methods include addition calculation instead of multiplication calculation, the Winograd fast convolution algorithm, and the Im2col convolution acceleration algorithm, among others.

4.1.1. Fixed-Point Quantitative

Since the bit width of the number involved in the operation is finite and constant in FPGA calculations, the floating-point decimal number involved in the operation needs to be fixed to limit the bit width of a floating-point decimal to the range of bit width allowed by FPGA. Floating-point decimals mean that the decimal point position is not fixed, and fixed-point decimals mean that the decimal point position is fixed. Fixed-point quantization is the use of finite digits to represent infinite precision numbers, that is, the quantization function maps the full precision numbers (the activation parameters, weight parameters, and even gradient values) to a finite integer space. Fixed-point quantization can greatly reduce the memory space of each parameter and the computational complexity within the acceptable accuracy loss range, so as to achieve neural network acceleration.

In 2011, Vincent Vanhoucke first proposed the linear fixed-point 8-bit quantization technology, which was initially used in X86 CPU, greatly reducing the computational cost, with it then being slowly applied to FPGA [55]. In March 2020, Shiguang Zhang et al. [56] proposed a reconfigurable CNN accelerator with an AXI bus based on advanced RISC machine (ARM) + FPGA architecture.

In June of the same year, Zongling Li et al. [57] designed a CNN weight parameter quantization method suitable for FPGA, different from the direct quantization operation in the literature [56]. They transformed the weight parameter into logarithm base 2, which greatly reduced the quantization bit width, improved the quantization efficiency, and reduced the delay. Using a shift operation instead of a convolution multiplication operation saves a large number of computational resources.

4.1.2. Multiplication Optimization

Matrix operations mainly appear in the training and forward computation of neural networks and they occupy a dominant position; thus, it is of great significance to accelerate matrix operation. The optimization of matrix multiplication can be achieved by reducing the number of multiplication computations and increasing the computation speed. Matrix operations often have remarkable parallelism. Scholars usually use the characteristics of FPGA parallel computing to optimize matrix operations.

As shown in **Figure 7**, the architecture uses specialized compressed interleave sparse row (CISR) coding to efficiently process multiple rows of the matrix in parallel, combined with a caching design that eliminates the replication of the buffer carrier and enables larger vectors to be stored on the chip. The design maximizes the bandwidth utilization by organizing the data from memory into parallel channels, which can keep the hardware complexity low while greatly improving the parallelism and speed of data processing.

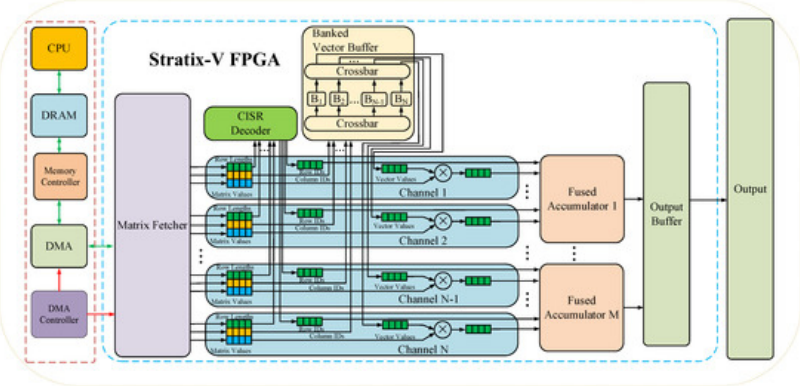

Figure 7. Neural network optimization technology based on FPGA.

4.1.3. Convolution Optimization

For the optimization of convolution algorithms, scholars have provided the following two ideas. One is the Winograd fast convolution algorithm, which speeds up convolution by increasing addition and reducing multiplication. Another is the Im2col convolution optimization algorithm that sacrifices storage space in order to improve the speed of the convolution operation.

4.1.4. Pipelined Design

Pipelined design is a method of systematically dividing combinational logic, inserting registers between the parts (hierarchies), and temporarily storing intermediate data. The purpose is to decompose a large operation into a number of small operations. Each small operation takes less time, shortens the length of the path that a given signal must pass in a clock cycle, and improves the frequency of operations; moreover, small operations can be executed in parallel and can improve the data throughput.

The design method of Pipeline can greatly improve the working speed of the system. This is a typical design approach that converts “area” into “velocity”. The “area” here mainly refers to the number of FPGA logic resources occupied by the design, which is measured by the consumed flip-flop (FF) and look-up table (LUT). “Speed” refers to the highest frequency that can be achieved while running stably on the chip. The two indexes of area and speed always run through the design of FPGA, which is the final standard of design quality evaluation. This method can be widely used in all kinds of designs, especially the design of large systems with higher speed requirements. Although pipelining can increase the use of resources, it can reduce the propagation delay between registers and ensure that the system maintains a high system clock speed. In the convolutional layer of deep neural networks, when two adjacent convolutional iterations are performed and there is no data dependency, pipelining allows for the next convolutional operation to start before the current operation is completely finished, improving the computational power, with pipelining having become a necessary operation for most computational engine designs. In practical application, considering the use of resources and the requirements of speed, the series of pipelines can be selected according to the actual situation in order to meet the design needs.

4.2. Bandwidth Optimization

In the deployment of neural networks, all kinds of neural networks must rely on specific computing platforms (such as CPU/GPU/ASIC/FPGA) in order to complete the corresponding algorithms and functions. At this point, the “level of compatibility” between the model and the computing platform will determine the actual performance of the model. Samuel Williams et al. [87] proposed an easy-to-understand visual performance model, the roof-line model, and proposed a quantitative analysis method using operational intensity. The formula showcasing that the model can reach the upper limit of the theoretical calculation performance on the computing platform is provided. It offers insights for programmers and architects in order to improve the parallelism of floating-point computing on software and hardware platforms and is widely used by scholars to evaluate their designs on various hardware platforms to achieve better design results.

The roof-line model is used to measure the maximum floating point computing speed that the model can achieve within the limits of a computing platform. Computing power and bandwidth are usually used to measure performance on computing platforms. Computing power is also known as the platform performance ceiling, which refers to the number of floating pointed operations per second that can be completed by a computing platform at its best, in terms of FLOP/s or GLOP/s. Bandwidth is the maximum bandwidth of a computing platform. It refers to the amount of memory exchanged per second (byte/s or GB/s) that can be completed by a computing platform at its best. Correspondingly, the computing intensity limit Imax is the computing power divided by the bandwidth. It describes the maximum number of calculations per unit of memory exchanged on the computing platform in terms of FLOP/byte.

In recent years, many application achievements based on the roof-line model are also involved in the optimization design of neural networks that are based on FPGA. In 2020, Marco Siracusa et al. [88] focused on the optimization of high-level synthesis (HLS). They found that although advanced synthesis (HLS) provides a convenient way to write FPGA code in a general high-level language, it requires a large amount of effort and expertise to optimize the final FPGA design of the underlying hardware, and therefore they proposed a semi-automatic performance optimization method that was based on the FPGA-based hierarchical roof line model. By combining the FPGA roof-line model with the Design Space Exploration (DSE) engine, this method is able to guide the optimization of memory limit and can automatically optimize the design of a calculation limit, and thus the workload is greatly reduced, and a certain acceleration performance can be obtained.

4.3. Memory and Access Optimization

Data storage and access is an essential part of neural networks. A large amount of data will occupy limited memory space. At the same time, a large amount of memory access will greatly increase the execution time of network models and reduce the computational efficiency. In the aspect of memory and access optimization, ping-pong cache, data reuse, standard data access, and so on are usually used for optimization.

4.3.1. Ping-Pong Cache

The ping-pong cache is a commonly used data flow control technique, which is used to allocate the input data flow to two random access memory (RAM) buffers in equal time through the input data selection unit and realize the flow transmission of data by switching between the two RAM reads and writes.

As shown in **Figure 8**, a~c describes the complete operation process of completing the cache of data in RAM A and B and the output of data. Each letter represents the input and output of data at the same time, the inward arrow represents the cache of data, and the outward arrow represents the output data. The specific caching process is as follows: The input data stream is allocated to two data buffers in equal time through the input data selection unit, and the data buffer module is generally RAM. In the first buffer cycle, the input data stream is cached to the data buffer module RAM A. In the second buffer cycle, the input data stream is cached to the data buffer module RAM B by switching the input data selection unit, and the first cycle data cached by RAM A is transmitted to the output data selection unit. In the third buffer cycle, the input data stream is cached to RAM A, and the data cached by RAM B in the second cycle is passed to the output data selection unit through another switch of the input data selection unit.

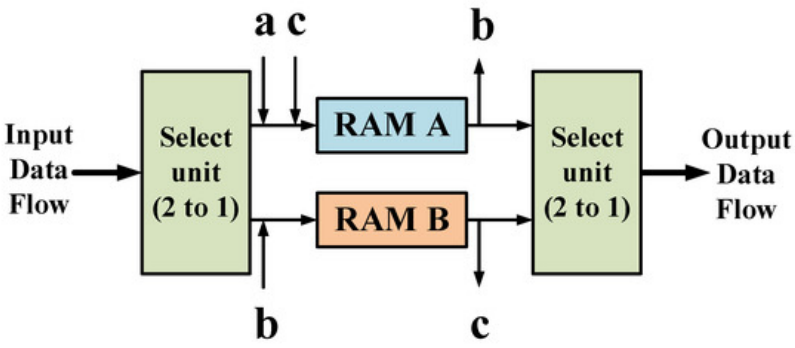

Figure 8. Ping-pong cache description.

4.3.2. Data Reuse

Data reuse is simply the reuse of data. As can be seen from the roof-line model, when the data reuse rate was low, bandwidth became the bottleneck affecting performance, and the computing resources of FPGA were not fully utilized. Therefore, through data reuse, the actual application bandwidth of memory is greater than the theoretical bandwidth, which increases the upper limit of the bandwidth and also reduces the storage pressure of memory and the amount of data cache and data exchange, so as to reduce the unnecessary time spent.

As shown in **Figure 9**, in the data operation of neural networks, data reuse is usually divided into input feature map reuse, filter reuse, and convolutional reuse. An input feature map reuse is the reuse of the same input and replacing it with the next input after all the convolution kernels are computed. A filter reuse means that if there is a batch of inputs, the same convolution kernel for the batch is reused and the data are replaced after all the inputs in the batch are calculated. Convolutional reuse uses the natural calculation mode in the convolutional layer and uses the same convolution kernel to calculate the output of different input map positions. In general, these three data reuse modes reuse data at different stages in order to achieve the purpose of increasing performance. Of course, data reuse can also be divided by time reuse and space reuse. For example, if the data in a small buffer is repeatedly used in multiple calculations, it is called time reuse. If the same data are broadcast to multiple PEs for simultaneous calculation, it is called space reuse.

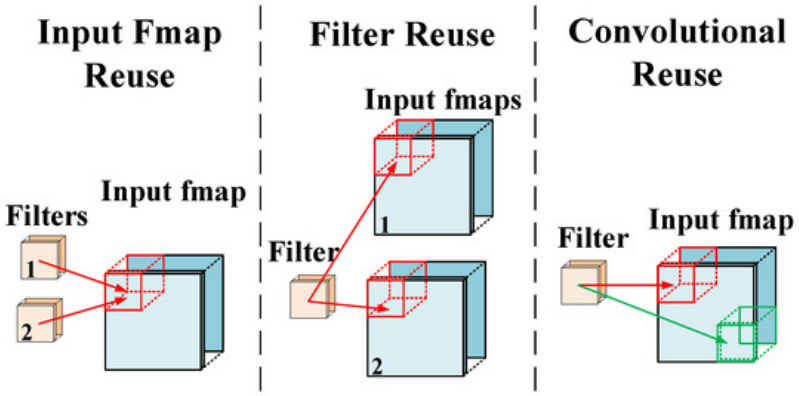

Figure 9. Common data reuse modes.

4.3.3. Standardized Data Access and Storage

A large amount of data operation and frequent data access are the problems that neural networks must encounter when they are deployed on portable systems such as FPGA. In the optimization design of neural networks that are based on FPGA, FPGA is usually used as a coprocessor, that is, the CPU writes the instructions to the memory, and then the FPGA reads and executes the instructions from the memory unit, and following this, writes the calculation results to the memory. Therefore, by standardizing data access, the read and write efficiency of data can be improved, and the actual bandwidth upper limit can be increased.

5. Design of the DNN Accelerator and Acceleration Framework Based on FPGA

In the optimization design of neural networks that are based on FPGA, some FPGA synthesis tools are generally used. The existing synthesis tools (HLS, OpenCL, etc.) that are highly suitable for FPGA greatly reduce the design and deployment time of neural networks, and the hardware-level design (such as RTL, register transfer level) can improve the efficiency and achieve a better acceleration effect. However, with the continuous development of neural networks, its deployment on FPGA has gradually become the focus of researchers. This further accelerates the emergence of more accelerators and acceleration frameworks for neural network deployment on FPGA. This is because with the acceleration of the specific neural network model, the idea is the most direct, and the design purpose is also the clearest. These accelerators are often hardware designs for the comprehensive application of the various acceleration techniques described above. When used in specific situations, such accelerators usually only need to fine-tune the program or parameters to be used, which is very convenient [13].

5.1. FPGA Accelerator Design for Different Neural Networks

With the continuous development of deep learning, neural networks have achieved great success in various fields, such as image, speech, and video, and neural networks are developing towards deeper and more complex network design. The way in which to deploy more complex neural networks on FPGA and meet certain speed requirements has become the focus of researchers. In the existing research, a large number of neural network accelerator designs that are based on FPGA have emerged. The main one is the CNN accelerator that is based on FPGA, the RNN accelerator based on FPGA, and the GAN accelerator based on FPGA.

5.1.1. The CNN Accelerator Based on FPGA

In the design of many accelerators for FPGA-accelerated convolutional neural networks, most of them focus on improving the network computing power, data transmission speed, and memory occupancy. Scholars usually improve the parallel

computing ability of convolutional neural networks in order to improve the computational efficiency and reduce the amount of data and data access in order to solve the problem of large data transmission overhead and large data volume.

In order to solve the problem of heavy computational workload, which limits the deployment of deep convolutional neural networks on embedded devices, the hardware structure shown in **Figure 10** was designed in [100]. In this hardware architecture, all buffers use the ping-pong data transfer mechanism to mask the data transfer time with the computation time to improve the performance.

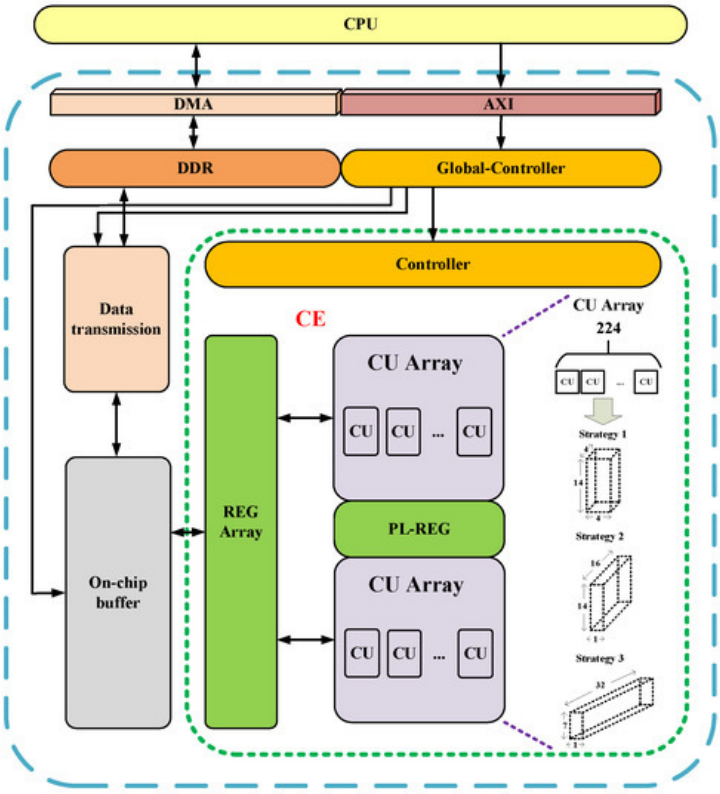

Figure 10. Accelerator architecture based on a high-speed computing engine (CE) [100].

The computing engine (CE) is mainly composed of two computing unit (CU) arrays and several register arrays. Inside each computing unit is a “tree” structure of the bottom multiplier combined with the multilayer adder. Each CU array has 224 CU and can be configured to compute, in parallel, three different parallel modes of output feature maps of the dimension 4 × 14 × 4, 16 × 14 × 1, or 32 × 7 × 1. The flexible configuration ensures high CU utilization in different convolution parameters, so as to maintain a high operation speed. The register array includes the input feature map register (I-REG), the weight register (W-REG), the partial sum register (PS-REG), the output feature map register (O-REG), and the pooling register (PL-REG). On-chip buffers include buffers for input feature map (IBUF), weight (WBUF), partial sum (PBUF), and output feature map (OBUF). Each buffer consists of multiple blocks of RAM, which allow more data to be read or written simultaneously to improve read/write efficiency.

In this framework, the image and weight data were first added to the double data rate SDRAM (DDR) by the CPU-controlled direct memory access (DMA), and the top controller of the accelerator was configured, through which the control instructions were assigned to each module. The data transfer module reads the input data and weight parameters from the DDR to the on-chip buffer. Then, the computing engine (CE) performs the convolution operation, and the final result is written back to the DDR through the data transmission module. The architecture successfully deployed VGG-16, ResNet50, and YOLOv2-Tiny lightweight networks on a Xilinx ZynQ-7 ZC706 evaluation version operating at 200 MHz. The performances of 163 GOPS and 0.36 GOPS/DSP were achieved on VGG-16 with only 448 DSP resources.

Although this structure design achieves better performance and less resource consumption, on the whole, this structure does not consider the model parameter optimization of the fully connected layer, and the transmission overhead incurred when processing a large amount of data may limit the actual performance. In addition, this “tree” structure of a computing unit array can only process regular convolution operations. Although it can be configured into three different parallel modes in order to enhance the adaptability of the convolution layer to a certain extent, it is still not suitable for the convolutional neural network after sparse processing. Therefore, with the application of increasingly more irregular convolution operations, the usage scenarios of this hardware structure may be limited.

5.1.2. The RNN Accelerator Based on FPGA

Because the traditional RNN has the problem of gradient disappearance and explosion, the network performance is not good in the application, requiring long-term input information. Some researchers proposed a variant of RNN, long short-term memory (LSTM), to solve this problem. Although LSTM can solve the gradient disappearance problem, it cannot avoid the gradient explosion problem, and because it introduces many gating units, it leads to the problem of a large number of parameters [101]. Others have proposed the gate recurrent unit (GRU), which, like LSTM, has been put forward to solve problems such as long-term memory and gradients in back propagation [102]. In many cases, the GRU and LSTM are practically similar in performance. The largest difference between GRU and LSTM is that GRU combines the forgetting gate and the input gate into one “update gate,” and the network does not provide additional memory states. Instead, the output results are continuously recycled back as memory states, and the input and output of the network become particularly simple. However, GRU still cannot completely solve the problem of vanishing gradient.

In order to reduce RNN memory overhead for acoustic task, the [85] used a quantitative method to reduce the activation volume, wherein the time-consuming floating-point arithmetic was replaced by the faster floating-point arithmetic, greatly improving the network operation speed and joining the parameters of the mechanism of sharing and pipelined design in order to increase parallelism and reduce storage, further improving the network throughput and reducing the delay. However, the data quantized by this scheme is still the floating point. Fixed-point quantization can greatly improve the speed of data operation and reduce the memory overhead without much impact on the accuracy. Due to the feedback dependence of LSTM, high parallelism cannot be achieved on general processors such as the CPU and GPU.

The accelerator architecture firstly divides the input matrix and weight matrix into small blocks in advance, increases the parallelism of the matrix operation, and combines the pulsating array algorithm in order to accelerate the computation. Moreover, the problem of low data reading efficiency caused by data discontinuity during sequential data reading is solved by rearranging the elements in the matrix. Among them, the traditional processing element (PE) from memory read data performs various calculations and then writes the results back to the storage architecture, wherein the pulsating array appears in the form of lines, and each PE calculation no longer relies on memory access, only the first array PE in terms of reading data from memory, after processing the results directly to the next PE.

There are also some studies devoted to the compression of neural networks. Reference [103] proposed a hybrid iterative compression algorithm (HIC) for LSTM/GRU to compress LSTM/GRU networks. The utilization of block RAM (BRAM) is improved by rearranging the weights of the data stream of the matrix operation unit based on block structure matrix (MOU-S) and by fine-grained parallel configuration of matrix vector multiplication (MVM). At the same time, the combination of quantization and pruning greatly reduces the storage pressure.

5.1.3. The GAN Accelerator Based on FPGA

GANs mainly consist of a generator and a discriminator, producing better output through mutual game learning between generator G and discriminator D. In the original GAN theory, both G and D are not required to be neural networks, as long as the corresponding functions that are generated and discriminated against can be fitted. However, in practical application, deep neural networks used as G and D generally, so in the GAN accelerator design based on FPGA, it is basically aimed at the generator and the discriminator to optimize. Therefore, it is essentially optimized for CNN, RNN, or other neural network, mainly from the lower network number, wherein the accelerators are designed from the perspectives of reducing storage pressure, reducing computational complexity, and improving network computing speed.

The acceleration framework is shown in **Figure 11**. Firstly, the cascade fast FIR (finite impulse response) algorithm (CFFA) is optimized for GAN training, and the fast convolution processing element (FCPE) based on the optimization algorithm is introduced to support various computing modes during GAN training. In the input prefetcher module and weight prefetcher module, 16-bit fixed point and 8-bit fixed point were used, respectively, to process data, greatly improving the data processing speed. Finally, the architecture achieved 315.18 GOPS performance and 83.87 GOPS/W energy efficiency on the Xilinx VCU108 FPGA platform with 200 MHz operating frequency. Experiments showed that the architecture was able to achieve high energy efficiency with less FPGA resources. Although the architecture can effectively avoid the large consumption of hardware resources by cascading small parallel FIR structures to larger parallel FIR structures, this operation greatly increases the delay. When the number of cascades reaches a certain limit, the performance loss caused by the delay is incalculable.

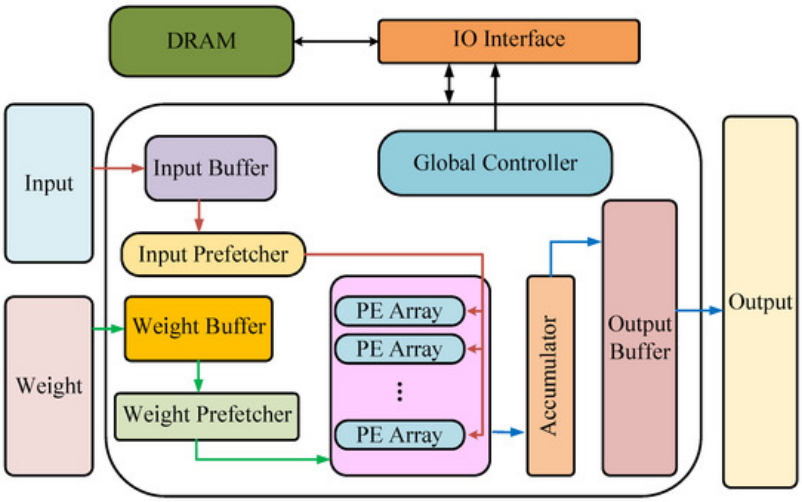

Figure 11. The GAN acceleration framework.

5.2. Accelerator Design for Specific Application Scenarios

In the practical application of neural networks, people often customize FPGA accelerators with required functions according to specific application scenarios. Since accelerators customized according to specific application scenarios are relatively easy to design and can effectively solve the corresponding problems, this method is commonly used to accelerate neural networks in specific application scenarios, especially in speech recognition, image processing, natural language processing, and other fields.

5.2.1. FPGA Accelerator for Speech Recognition

Speech recognition is a technology that enables machines to automatically recognize and understand human spoken language through speech signal processing and pattern recognition. In short, it is the technology that allows a machine to transform a speech signal into a corresponding text or command through the process of recognition and understanding. Speech recognition has been applied in many fields, including speech recognition translation, voice paging and answering platforms, independent advertising platforms, and intelligent customer service. Speech recognition has strong real-time performance and high delay requirements, and thus it generally relies on the FPGA platform.

In terms of accelerating the application of speech recognition, the authors of [104] focused on the preprocessing stage of speech signals and proposed the use of a GAN to enhance speech signals and reduce noise in speech signals, so as to improve speech quality and facilitate speech recognition and processing. In order to reduce energy consumption and improve the speed of speech recognition, the authors of [105] proposed an FPGA accelerator structure called balanced row dual-ratio sparsity inducing pruning algorithm (BRDS) for speech recognition, as shown in **Figure 12**. The accelerator compresses the LSTM network by pruning algorithm in order to reduce computational complexity and uses data reuse and pipeline design to achieve low power consumption and low delay. However, the acceleration architecture does not consider the problem of multi-core parallel load imbalance after LSTM model compression, which may occur in the actual use, thus affecting the performance of the whole accelerator.

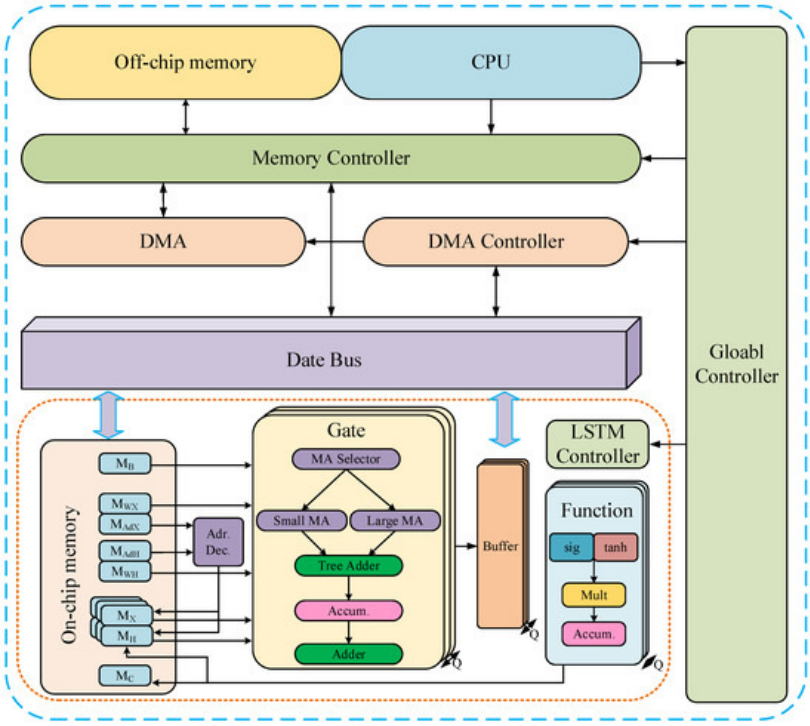

Figure 12. BRDS: FPGA accelerator for speech recognition.

5.2.2. FPGA Accelerator for Speech Recognition

Image processing refers to the process of extracting, analyzing, and processing image information by computer, which is one of the earliest application fields of deep learning. Image processing techniques include point processing, group processing, geometric processing, and frame processing. The main research contents include image enhancement, image restoration, image recognition, image coding, image segmentation, among others. Since 2012, ImageNet competition, image recognition, and processing technology have been widely used in image classification, face recognition, and other fields. In the design of an FPGA accelerator for image processing, the accelerator is designed mainly for the purpose of reducing the amount of data, memory, and computation demand [81][106][107][108].

5.2.3. FPGA Accelerator for Natural Language Processing

Natural language processing (NLP) is a technology that uses the natural language used by humans to communicate with machines. It studies various theories and methods that can realize effective communication between humans and computers using natural language, and it is also one of the important application fields of deep learning.

In terms of accelerating NLP, the authors of [109] note that although converter-based language representations [110] have reached state-of-the-art accuracy on a variety of NLP tasks, the deployment of large model networks has been a challenge for resource-constrained computing platforms. Weight pruning is used to reduce the weight parameters, which can effectively compress the network model and improve the network performance. In order to solve the problem of large storage requirements and decreased network performance caused by a large number of parameters when RNN is applied to natural language processing, the authors [111] designed an FPGA accelerator based on fixed-point quantization technology to meet the computing resource requirements of RNN, reducing memory consumption by 90%. And the accuracy loss is less than 1%.

5.3. FPGA accelerator for optimization strategy

In the design of FPGA-based neural network accelerators, it is often necessary to study the characteristics of the neural network and carry out targeted optimization according to its operating characteristics, among which a reasonable optimization strategy can significantly improve the performance and resource utilization of the accelerator, which is a common optimization strategy for optimization calculation, storage optimization, and other improvements.

5.3.1. Calculation optimization

In neural networks, convolutional layers and convolutional operations are often the focus, especially in convolutional neural networks applied to image processing. Convolutional operations are more intensive, processing time is longer, and resource consumption is higher, which limits the deployment of convolutional neural networks in edge platforms such as FPGA.

The usual optimization direction is to increase the parallelism between different layers of the neural network model, the parallelism between different output feature maps, the parallelism between different pixels, and the parallelism between different pixels in order to shorten the processing time and improve the network performance. It can also use loop flows and loop unwrapping to build deeper pipelines to shorten the overall execution time of the network and reduce time overhead. The matrix cycle is divided into smaller modules by the cyclic block technique, and each module can be calculated in parallel to improve computational parallelism.

5.3.2. Storage Optimization

Due to the large amount of data and parameters in the neural network, the problem of parameter and data storage is inevitable when deploying the neural network. In the optimization of storage, the strategy of reducing the accuracy of data is usually adopted to reduce the amount of data, so as to improve the operation speed. Data reuse methods can be adopted to greatly reduce data storage and memory space occupancy.

5.4. Other FPGA accelerator design

In addition to the above-mentioned accelerator designs, there are other accelerator designs, such as those based on hardware templates. By using off-the-shelf hardware templates to design accelerators that require only improvements to the module and configuration parameters based on the specific problem, the deployment of the model is greatly improved [112].

References

1. Subramanian, A.S.; Weng, C.; Watanabe, S.; Yu, M.; Yu, D. Deep learning based multi-source localization with source splitting and its effectiveness in multi-talker speech recognition. Comput. Speech Lang. 2022, 75, 101360.
2. Kumar, L.A.; Renuka, D.K.; Rose, S.L.; Shunmuga-priya, M.C.; Wartana, I.M. Deep learning based assistive technology on audio visual speech recognition for hearing impaired. Int. J. Cogn. Comput. Eng. 2022, 3, 24–30.
3. Roßbach, J.; Kollmeier, B.; Meyer, B.T. A model of speech recognition for hearing-impaired listeners based on deep learning. J. Acoust. Soc. Am. 2022, 151, 1417–1427.
4. Garcia, G.R.; Michau, G.; Ducoffe, M.; Gupta, J.S.; Fink, O. Temporal signals to images: Monitoring the condition of industrial assets with deep learning image processing algorithms. Proc. Inst. Mech. Eng. Part O J. Risk Reliab. 2022, 236, 617–627.
5. Suganyadevi, S.; Seethalakshmi, V.; Balasamy, K. A review on deep learning in medical image analysis. Int. J. Multimed. Inf. Retr. 2022, 11, 19–38.
6. Zuo, C.; Qian, J.; Feng, S.; Yin, W.; Li, Y.; Fan, P.; Han, J.; Qian, K.; Qian, C. Deep learning in optical metrology: A review. Light Sci. Appl. 2022, 11, 1–54.
7. Lauriola, I.; Lavelli, A.; Aiolli, F. An introduction to deep learning in natural language processing: Models, techniques, and tools. Neurocomputing 2022, 470, 443–456.
8. Razumovskaia, E.; Glavaš, G.; Majewska, O.; Ponti, E.M.; Vulic, I. Natural Language Processing for Multilingual Task-Oriented Dialogue. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, Dublin, Ireland, 22–27 May 2022; pp. 44–50.
9. Li, B.; Hou, Y.; Che, W. Data Augmentation Approaches in Natural Language Processing: A Survey; AI Open: Beijing, China, 2022.
10. Hu, Y.; Liu, Y.; Liu, Z. A Survey on Convolutional Neural Network Accelerators: GPU, FPGA and ASIC. In Proceedings of the 2022 14th International Conference on Computer Research and Development (ICCRD), Shenzhen, China, 7–9 January 2022; pp. 100–107.
11. Mittal, S.; Umesh, S. A survey on hardware accelerators and optimization techniques for RNNs. J. Syst. Archit. 2021, 112, 101839.
12. Shrivastava, N.; Hanif, M.A.; Mittal, S.; Sarangi, S.R.; Shafique, M. A survey of hardware architectures for generative adversarial networks. J. Syst. Archit. 2021, 118, 102227.
13. Liu, T.; Zhu, J.; Zhang, Y. Review on FPGA-Based Accelerators in Deep Learning. J. Front. Comput. Sci. Technol. 2021, 15, 2093–2104.
14. Jiao, L.; Sun, Q.; Yang, Y.; Feng, Y.; Li, X. Development, Implementation and Prospect of FPGA-Based Deep Neural Networks. Chin. J. Comput. 2022, 45, 441–471.

15. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 1943, 5, 115–133.
16. Turing, A. Intelligent Machinery (1948); B. Jack Copeland: Oxford, NY, USA, 2004; p. 395.
17. Hebb, D.O. The Organization of Behavior: A Neuropsychological Theory; Psychology Press: Mahwah, NJ, USA; London, UK, 2005.
18. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. 1958, 65, 386.
19. Minsky, M.; Papert, S.A. Perceptrons, Reissue of the 1988 Expanded Edition with a New Foreword by Léon Bottou: An Introduction to Computational Geometry; MIT Press: Cambridge, MA, USA, 2017.
20. Werbos, P.J. Backpropagation through time: What it does and how to do it. Proc. IEEE 1990, 78, 1550–1560.
21. Fukushima, K.; Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and Cooperation in Neural Nets; Springer: Berlin/Heidelberg, Germany, 1982; pp. 267–285.
22. Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA 1982, 79, 2554–2558.
23. Ackley, D.H.; Hinton, G.E.; Sejnowski, T.J. A learning algorithm for Boltzmann machines. Cogn. Sci. 1985, 9, 147–169.
24. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. Nature 1986, 323, 533–536.
25. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1989, 1, 541–551.
26. Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. Science 2006, 313, 504–507.
27. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. Commun. ACM 2017, 60, 84–90.
28. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
29. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv 2014, arXiv:1409.1556.
30. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. Commun. ACM 2020, 63, 139–144.
31. Sun, Y.; Wang, X.; Tang, X. Deep learning face representation from predicting 10,000 classes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, 23–28 June 2014; pp. 1891–1898.
32. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, 23–28 June 2014; pp. 580–587.
33. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
34. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
35. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. arXiv 2018, arXiv:1804.02767.
36. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. arXiv 2020, arXiv:2004.10934.
37. Li, C.; Li, L.; Jiang, H.; Wenig, K.; Geng, Y.; Li, L.; Ke, Z.; Li, Q.; Cheng, M.; Nie, W.; et al. YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. arXiv 2022, arXiv:2209.02976.
38. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv 2022, arXiv:2207.02696.
39. Guo, W.; Xu, G.; Liu, B.; Wang, Y. Hyperspectral Image Classification Using CNN-Enhanced Multi-Level Haar Wavelet Features Fusion Network. IEEE Geosci. Remote Sens. Lett. 2022, 19, 1–5.

40. Chakraborty, S.; Paul, S.; Hasan, K.M. A transfer learning-based approach with deep cnn for covid-19-and pneumonia-affected chest x-ray image classification. SN Comput. Sci. 2022, 3, 1–10.
41. Sharma, T.; Nair, R.; Gomathi, S. Breast cancer image classification using transfer learning and convolutional neural network. Int. J. Mod. Res. 2022, 2, 8–16. Available online: http://ijmore.co.in/index.php/ijmore/article/view/6 (accessed on 23 September 2022).
42. Han, G.; Huang, S.; Ma, J.; He, Y. Meta faster r-cnn: Towards accurate few-shot object detection with attentive feature alignment. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 1 March–22 February 2022; Volume 36, pp. 780–789.
43. Ramachandra, A.C. Real Time Object Detection System with YOLO and CNN Models: A Review. arXiv 2022, arXiv:2208.00773.
44. Saralioglu, E.; Gungor, O. Semantic segmentation of land cover from high resolution multispectral satellite images by spectral-spatial convolutional neural network. Geocarto Int. 2022, 37, 657–677.
45. Valdez-Rodríguez, J.E.; Calvo, H.; Felipe-Riverón, E.; Moreno-Armendariz, M.A. Improving Depth Estimation by Embedding Semantic Segmentation: A Hybrid CNN Model. Sensors 2022, 22, 1669.
46. Nguyen, C.; Asad, Z.; Deng, R.; Huo, Y. Evaluating transformer-based semantic segmentation networks for pathological image segmentation. In Proceedings of the Medical Imaging 2022: Image Processing, Tianjin, China, 14–16 January 2022; Volume 12032, pp. 942–947.
47. Sağlam, S.; Tat, F.; Bayar, S. FPGA Implementation of CNN Algorithm for Detecting Malaria Diseased Blood Cells. In Proceedings of the 2019 International Symposium on Advanced Electrical and Communication Technologies (ISAECT), Rome, Italy, 27–29 November 2019; pp. 1–5.
48. Zhang, Q. Application of CNN Optimization Design Based on APSOC in the Classification of Congenital Heart Disease. Master's Thesis, Yunnan University, Kunming, China, 2020.
49. Wang, C. Implementation and Verification of CNN Based on FPGA. Ph.D. Thesis, Hebei University, Baoding, China, 2020.
50. Ferreira, J.C.; Fonseca, J. An FPGA implementation of a long short-term memory neural network. In Proceedings of the 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 30 November–2 December 2016; pp. 1–8.
51. Guan, Y.; Yuan, Z.; Sun, G.; Cong, J. FPGA-based accelerator for long short-term memory recurrent neural networks. In Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, Japan, 16–19 January 2017; pp. 629–634.
52. Li, Z.; Ding, C.; Wang, S.; Wen, W.; Zhou, Y.; Liu, C.; Qiu, Q.; Xu, W.; Lin, X.; Qian, X.; et al. E-RNN: Design optimization for efficient recurrent neural networks in FPGAs. In Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 16–20 February 2019; pp. 69–80.
53. Gao, C.; Rios-Navarro, A.; Chen, X.; Liu, S.C.; Delbruck, T. EdgeDRNN: Recurrent neural network accelerator for edge inference. IEEE J. Emerg. Sel. Top. Circuits Syst. 2020, 10, 419–432.
54. Yazdanbakhsh, A.; Brzozowski, M.; Khaleghi, B.; Ghodrati, S.; Samadi, K.; Kim, N.S.; Esmaeilzadeh, H. Flexigan: An end-to-end solution for fpga acceleration of generative adversarial networks. In Proceedings of the 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Boulder, CO, USA, 29 April–1 May 2018; pp. 65–72.
55. Vanhoucke, V.; Senior, A.; Mao, M.Z. Improving the Speed of Neural Networks on CPUs. In Proceedings of the Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011, Granada, Spain, 15 December 2011.
56. Zhang, S.; Cao, J.; Zhang, Q.; Zhang, Q.; Zhang, Y.; Wang, Y. An fpga-based reconfigurable cnn accelerator for yolo. In Proceedings of the 2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, 8–11 May 2020; pp. 74–78.
57. Li, Z.; Chen, J.; Wang, L.; Cheng, B.; Yu, J.; Jiang, S. CNN Weight Parameter Quantization Method for FPGA. In Proceedings of the 2020 IEEE 6th International Conference on Computer and Communications (ICCC), Chengdu, China, 11–14 December 2020; pp. 1548–1553.
58. Chang, S.E.; Li, Y.; Sun, M.; Shi, R.; So, H.K.H.; Qian, X.; Wang, Y.; Lin, X. Mix and match: A novel fpga-centric deep neural network quantization framework. In Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea, 27 February–3 March 2021; pp. 208–220.
59. Zhao, X.; Wang, Y.; Cai, X.; Liu, C.; Zhang, L. Linear Symmetric Quantization of Neural Networks for Low-Precision Integer Hardware. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia,

30 April 2020.

60. Bao, Z.; Zhan, K.; Zhang, W.; Guo, J. LSFQ: A Low Precision Full Integer Quantization for High-Performance FPGA-Based CNN Acceleration. In Proceedings of the 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), Tokyo, Japan, 14–16 April 2021; pp. 1–6.
61. Zhao, X.; Zhang, X.; Yang, F.; Xu, P.; Li, W.; Chen, F. Research on Machine Learning Optimization Algorithm of CNN for FPGA Architecture. J. Phys. Conf. Ser. 2021, 2006, 012012.
62. Shi, T.J.; Liu, Y.F.; Tian, J.; Zhao, Y.X. Design of FPGA recurrent neural network accelerator based on high level synthesis. Inform. Technol. Inform. 2022, 1, 151–153.
63. Fowers, J.; Ovtcharov, K.; Strauss, K.; Chung, E.S.; Sitt, G. A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication. In Proceedings of the 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, Boston, MA, USA, 11–13 May 2014; pp. 36–43.
64. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84.
65. Gupta, A.; Suneja, K. Hardware Design of Approximate Matrix Multiplier based on FPGA in Verilog. In Proceedings of the 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 13–15 May 2020; pp. 496–498.
66. Iakovidis, D.K.; Maroulis, D.E.; Bariamis, D.G. FPGA architecture for fast parallel computation of co-occurrence matrices. Microprocess. Microsyst. 2007, 31, 160–165.
67. Abbaszadeh, A.; Iakymchuk, T.; Bataller-Mompeán, M.; Francés-Villora, J.V.; Rosado-Muñoz, A. Anscalable matrix computing unit architecture for FPGA, and SCUMO user design interface. Electronics 2019, 8, 94.
68. Kala, S.; Nalesh, S. Efficient cnn accelerator on fpga. IETE J. Res. 2020, 66, 733–740.
69. Kang, S.; Lee, S.; Kim, B.; Kim, H.; Sohn, K.; Kim, N.S.; Lee, E. An FPGA-based RNN-T Inference Accelerator with PIM-HBM. In Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual, 27 February–1 March 2022; pp. 146–152.
70. Lavin, A.; Gray, S. Fast algorithms for convolutional neural networks. In Proceedings of the IEEE conference on computer vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4013–4021.
71. Lu, L.; Liang, Y. SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–29 June 2018; pp. 1–6.
72. Kala, S.; Mathew, J.; Jose, B.R.; Nalesh, S. UniWiG: Unified winograd-GEMM architecture for accelerating CNN on FPGAs. In Proceedings of the 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, India, 5–9 January 2019; pp. 209–214.
73. Bao, C.; Xie, T.; Feng, W.; Chang, L.; Yu, C. A power-efficient optimizing framework fpga accelerator based on winograd for yolo. IEEE Access 2020, 8, 94307–94317.
74. Wang, X.; Wang, C.; Cao, J.; Gong, L.; Zhou, X. Winonn: Optimizing fpga-based convolutional neural network accelerators using sparse winograd algorithm. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2020, 39, 4290–4302.
75. Li, B.; Qi, Y.R.; Zhou, Q.L. Design and optimization of target detection accelerator based on Winograd algorithm. Acta Electron. Sin. 2022, 50, 2387–2397.
76. Tang, F.; Zhang, W.; Tian, X.; Fan, X.; Cao, X. Optimization of Convolution Neural Network Algorithm Based on FPGA. ESTC 2017. Communications in Computer and Information Science; Springer: Singapore, 2018; Volume 857, pp. 131–140.
77. Yu, F.; Cao, Y.; Tang, Y. Realization of Quantized Neural Network for Super-resolution on PYNQ. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Fayetteville, AR, USA, 3–6 May 2020; p. 233.
78. Ye, T.; Kuppannagari, S.R.; Kannan, R.; Prasanna, V.K. Performance Modeling and FPGA Acceleration of Homomorphic Encrypted Convolution. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 30 August–3 September 2021; pp. 115–121.
79. Zhang, H.; Jiang, J.; Fu, Y.; Chang, Y.C. Yolov3-tiny Object Detection SoC Based on FPGA Platform. In Proceedings of the 2021 6th International Conference on Integrated Circuits and Microsystems (ICICM), Nanjing, China, 22–24 October 2021; pp. 291–294.

80. Xiao, C.; Shi, C.; Xu, D.; Lin, F.; Ning, K. SDST-Accelerating GEMM-based Convolution through Smart Data Stream Transformation. In Proceedings of the 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Taormina, Italy, 16–19 May 2022; pp. 396–403.
81. Özkilbaç, B.; Ozbek, I.Y.; Karacali, T. Real-Time Fixed-Point Hardware Accelerator of Convolutional Neural Network on FPGA Based. In Proceedings of the 2022 5th International Conference on Computing and Informatics (ICCI), New Cairo, Egypt, 9–10 March 2022; pp. 1–5.
82. Liu, Z.; Dou, Y.; Jiang, J.; Xu, J.; Li, S.; Zhou, Y.; Xu, Y. Throughput-optimized FPGA accelerator for deep convolutional neural networks. ACM Trans. Reconfigurable Technol. Syst. (TRETS) 2017, 10, 1–23.
83. Xing, Y.; Liang, S.; Sui, L.; Jia, X.; Qiu, J.; Liu, X.; Wang, Y.; Shan, Y.; Wang, Y. Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators. IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst. 2019, 39, 2668–2681.
84. Wang, W.; Zhou, K.; Wang, Y.; Wang, G.; Yang, Z.; Yuan, J. FPGA parallel structure design of convolutional neural network (CNN) algorithm. Microelectron. Comput. 2019, 36, 57–62, 66.
85. Wen, D.; Jiang, J.; Dou, Y.; Xu, J.; Xiao, T. An energy-efficient convolutional neural network accelerator for speech classification based on FPGA and quantization. CCF Trans. High Perform. Comput. 2021, 3, 4–16.
86. Varadharajan, S.K.; Nallasamy, V. P-SCADA-a novel area and energy efficient FPGA architectures for LSTM prediction of heart arrthymias in BIoT applications. Expert Syst. 2022, 39, e12687.
87. Williams, S.; Waterman, A.; Patterson, D. Roofline: An insightful visual performance model for multicore architectures. Commun. ACM 2009, 52, 65–76.
88. Siracusa, M.; Di-Tucci, L.; Rabozzi, M.; Williams, S.; Sozzo, E.D.; Santambrogio, M.D. A cad-based methodology to optimize hls code via the roofline model. In Proceedings of the 39th International Conference on Computer-Aided Design, Virtual, 2–5 November 2020; pp. 1–9.
89. Calore, E.; Schifano, S.F. Performance assessment of FPGAs as HPC accelerators using the FPGA Empirical Roofline. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 30 August–3 September 2021; pp. 83–90.
90. Feng, Y.X.; Hu, S.Q.; Li, X.M.; Yu, J.C. Implementation and optimisation of pulse compression algorithm on open CL-based FPGA. J. Eng. 2019, 2019, 7752–7754.
91. Di, X.; Yang, H.G.; Jia, Y.; Huang, Z.; Mao, N. Exploring efficient acceleration architecture for winograd-transformed transposed convolution of GANs on FPGAs. Electronics 2020, 9, 286.
92. Yu, X.L.; Li, B.Q.; Dong, M.S.; Yin, W.M. Target Detection and Tracking System Based on FPGA. In Proceedings of the IOP Conference Series: Materials Science and Engineering. IOP Publ. 2020, 793, 012008.
93. Li, T.Y.; Zhang, F.; Guo, W.; Shen, J.J.; Sun, M.Q. An FPGA-based JPEG preprocessing accelerator for image classification. J. Eng. 2022, 2022, 919–927.
94. Zhang, H.; Li, Z.; Yang, H.; Cheng, X.; Zeng, X. A High-Efficient and Configurable Hardware Accelerator for Convolutional Neural Network. In Proceedings of the 2021 IEEE 14th International Conference on ASIC (ASICON), Kunming, China, 26–29 October 2021; pp. 1–4.
95. Nguyen, X.Q.; Pham-Quoc, C. An FPGA-based Convolution IP Core for Deep Neural Networks Acceleration. REV J. Electron. Commun. 2022, 1, 1–2.
96. Dinelli, G.; Meoni, G.; Rapuano, E.; Pacini, T.; Fanucci, L. MEM-OPT: A scheduling and data re-use system to optimize on-chip memory usage for CNNs on-board FPGAs. IEEE J. Emerg. Sel. Top. Circuits Syst. 2020, 10, 335–347.
97. Miyajima, T.; Sano, K. A memory bandwidth improvement with memory space partitioning for single-precision floating-point FFT on Stratix 10 FPGA. In Proceedings of the 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 7–10 September 2021; pp. 787–790.
98. Zhang, B.; Zeng, H.; Prasanna, V. Accelerating large scale GCN inference on FPGA. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Fayetteville, AR, USA, 3–6 May 2020; p. 241.
99. Du, Z.; Zhang, Q.L.; Lin, M.; Li, S.; Li, X.; Ju, L. A comprehensive memory management framework for CPU-FPGA heterogenous SoCs. IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst. 2022; in press.
100. Li, X.; Huang, H.; Chen, T.; Gao, H.; Hu, X.; Xiong, X. A hardware-efficient computing engine for FPGA-based deep convolutional neural network accelerator. Microelectron. J. 2022, 128, 105547.
101. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. Neural Comput. 1997, 9, 1735–1780.

102. Cho, K.; Van-Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv 2014, arXiv:1406.1078.
103. Nan, G.; Wang, Z.; Wang, C.; Wu, B.; Wang, Z.; Liu, W.; Lombardi, F. An Energy Efficient Accelerator for Bidirectional Recurrent Neural Networks (BiRNNs) Using Hybrid-Iterative Compression with Error Sensitivity. IEEE Trans. Circuits Syst. I Regul. Pap. 2021, 68, 3707–3718.
104. Ram, S.R.; Kumar, M.V.; Subramanian, B.; Bacanin, N.; Zivkovic, M.; Strumberger, I. Speech enhancement through improvised conditional generative adversarial networks. Microprocess. Microsyst. 2020, 79, 103281.
105. Ghasemzadeh, S.A.; Tavakoli, E.B.; Kamal, M.; Afzali-Kusha, A.; Pedram, M. BRDS: An FPGA-based LSTM accelerator with row-balanced dual-ratio sparsification. arXiv 2021, arXiv:2101.02667.
106. Jiang, W.; Yu, H.; Ha, Y. A High-Throughput Full-Dataflow MobileNetv2 Accelerator on Edge FPGA. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2022; in press.
107. Zhang, F.; Li, Y.; Ye, Z. Apply Yolov4-Tiny on an FPGA-Based Accelerator of Convolutional Neural Network for Object Detection. In Proceedings of the Journal of Physics: Conference Series. IOP Publ. 2022, 2303, 012032.
108. Latotzke, C.; Ciesielski, T.; Gemmeke, T. Design of High-Throughput Mixed-Precision CNN Accelerators on FPGA. arXiv 2022, arXiv:2208.04854.
109. Peng, H.; Huang, S.; Geng, T.; Li, A.; Jiang, W.; Liu, H.; Wang, S.; Ding, C. Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning. In Proceedings of the 2021 22nd International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 7–9 April 2021; pp. 142–148.
110. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, 16–20 November 2020; pp. 38–45.
111. Rapuano, E.; Pacini, T.; Fanucci, L. A Post-training Quantization Method for the Design of Fixed-Point-Based FPGA/ASIC Hardware Accelerators for LSTM/GRU Algorithms. Comput. Intell. Neurosci. 2022, 2022, 9485933.
112. Li, Z.; Sun, M.; Lu, A.; Ma, H.; Yuan, G.; Xie, Y.; Tang, H.; Li, Y.; Leeser, M.; Wang, Z.; et al. Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization. arXiv 2022, arXiv:2208.05163.