A Runtime Programmable Accelerator for Convolutional and Multilayer Perceptron Neural Networks on FPGA

Ehsan Kabir[1(✉)], Arpan Poudel[1], Zeyad Aklah[2], Miaoqing Huang[1], and David Andrews[1]

[1] CSCE Department, University of Arkansas, Fayetteville, AR, USA
{ekabir,arpanp,mqhuang,dandrews}@uark.edu
[2] Computer Science Department, University of Thi-Qar, Nasiriyah, Iraq
zaklah@utq.edu.iq

Abstract. Deep neural networks (DNNs) are prevalent for many applications related to classification, prediction and regression. To perform different applications with better performance and accuracy, an optimized network architecture is required, which can be obtained through experiments and performance evaluation on different network topologies. However, a custom hardware accelerator is not scalable and it lacks the flexibility to switch from one topology to another at run time. In order to support convolutional neural networks (CNN) along with multilayer perceptron neural networks (MLPNN) of different sizes, we present in this paper an accelerator architecture for FPGAs that can be programmed during run time. This combined CNN and MLP accelerator (CNN-MLPA) can run any CNN and MLPNN applications without re-synthesis. Therefore, time spent on synthesis, placement and routing can be saved for executing different applications on the proposed architecture. Run time results show that the CNN-MLPA can be used for network topologies of different sizes without much degradation of performance. We evaluated the resource utilization and execution time on Xilinx Virtex 7 FPGA board for different benchmark datasets to demonstrate that our design is run time programmable, portable and scalable for any FPGA. The accelerator was then optimized to increase the throughput by applying pipelining and concurrency, and reduce resource consumption with fixed-point operations.

Keywords: FPGA · Neural network · MLP · CNN · Overlay · Flexible · Programmable · Reconfigurable · Accelerators · Custom hardware

1 Introduction

Deep neural networks have been applied to applications that are hard to solve using traditional rule based programming methods. A trained DNN for a

L. Gan et al. (Eds.): ARC 2022, LNCS 13569, pp. 32–46, 2022.
https://doi.org/10.1007/978-3-031-19983-7_3

particular application takes some input features and makes prediction, decision or classification. For many real time applications, a CPU based software system might not be fast enough to produce outputs as the size of the network grows for complex problems. Hence, hardware platforms like FPGA are used for DNN applications because of their massive parallel processing units to produce throughput higher than the CPU. The architecture of DNN with parallel inputs, outputs, and neurons in the hidden layers makes it possible. The reconfigurable logic blocks and interconnects, parallel memory and computing units, and low power consumption of FPGA have produced many FPGA accelerator [1,2]. Applications such as image compression, pattern recognition, signal processing, IoT device control, and biomedical applications (e.g., arrhythmia and epilileptic seizure detection etc.) are reported in [3,4]. Krisps et al. [5] showed how an artificial neural network (ANN) could be implemented within an FPGA for a real time hand detection and tracking system. Wayne et al. [6] designed a spiking neural network accelerator supporting large scale simulation on FPGA-based systems. Seul et. al. Several methods exist for compressing the size of data to reduce multiplication and addition operations for fast inference and to reduce hardware consumption. For example, [7,8] proposed binary neural network (BNN) inference engine on FPGAs for MNIST image classification with high accuracy. Convolutional neural network (CNN) [9] is used for applications such as image recognition, segmentation, speech recognition, medical diagnosis etc. Although applications with CNN is growing, MLP workloads still have a large share in open clouds operations by companies such as Facebook and Google [10,11]. However, these accelerators are customized for only one type of neural network used in a single application. Thus, the parameters for different network topologies need to be defined before synthesis for different applications. Moreover, creating custom DNN for different applications with hardware description language (HDL) or high level synthesis (HLS) code is an arduous and time consuming task. Therefore, a multi-purpose hardware accelerator is desirable to meet varied computational and memory requirements while supporting various neural networks for various applications. Some flexible and scalable accelerators for neural networks on FPGA have been reported in [12–14]. This paper presents such an accelerator for CNN and MLP applications.

The main contributions of this paper are:

- Designing a run time programmable hardware accelerator to run both CNN and MLPNN applications.
- Writing a parameterized high level synthesis code so that parameters such as number of processing elements (PEs), data representation (floating-point or fixed-point) and activation function (AF) implementation approach (BRAM lookup tables (LUTs) or synthesized logic-diffused multiplier) can be set before synthesis. This allows designers to adjust resource utilization by varying the parallel processing with PEs. It also enables changing the type of AFs and data precision.

- Making some parameters programmable (such as input size, output size, number of layers, neurons, channels and filters, size of filters and stride) so that they can be set (up to a maximum value) during run time. Thus, different topologies for different applications can be run without resynthesizing the hardware. Our experiments showed 50 h of reduction time on synthesis, placement, implementation and routing for maximum utilization on Virtex 7.
- Developing a switching technique to switch between CNN and MLP operations according to user's need. It enables the reuse of the same PEs for both MLP and CNN.
- Optimizing the accelerator to demonstrate some strategies that can be applied to make further improvement on the performance of the accelerator.
- Testing different benchmark data sets and network topologies to show programmable attributes and performance of the accelerator, and comparing them with Xilinx FINN and DPU framework for the same networks.

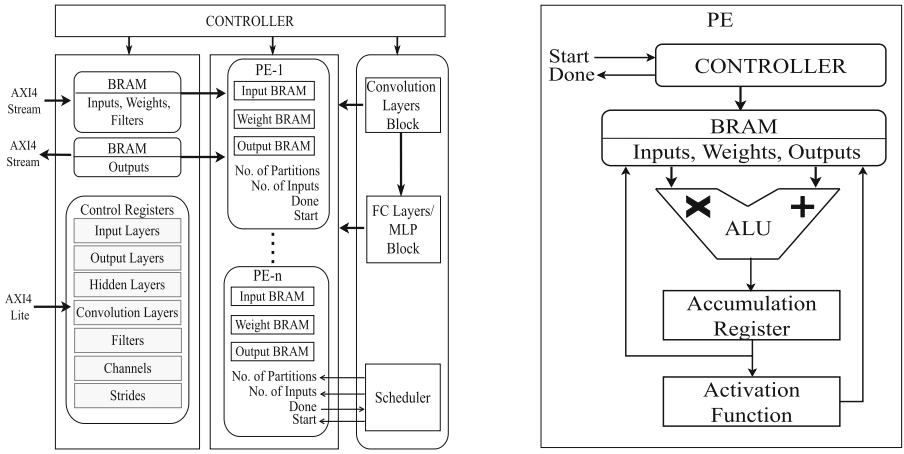
The rest of this paper is organized as follows. Section 2 introduces the CNN-MLPA architecture, and Sect. 3 presents the results for MLP. In Sect. 4, CNN feature of the accelerator is discussed in details along with its results. Finally, Sect. 5 concludes this paper.

2 CNN-MLPA Architecture

Figure 1a shows the generic structure of an CNN-MLP accelerator. It contains a 1D array of Processing Elements (PEs), a scheduler, a controller, configuration registers, local memory, and three external interface connections. Input data, weights for the filters in convolution layers (CLs), and weights in the fully connected (FC) layers are transferred through the same input channel. For a particular network, weights are fetched from DRAM according to their need and then they are stored in BRAM. The size of DRAM is the limitation for our design. The input and output interfaces are configured as FIFOs with a DMA engine (not shown) for fast transfers through the AXI-Stream interface. The AXI4-Lite interface is used to program the configuration registers and control the operation of the CNN-MLPA during run time. Users can switch between CNN and MLP during run time with a control signal. MLP is nothing but a fully connected neural network [15], whereas CNN contains convolution layers with multiple channels and filters [9,16] followed by the fully connected layers. Thus, for CNN applications, block of both convolution layers and MLPs are kept active; and for MLP applications, only the MLP block functions. The remainder of this section describes the functionality of different components of the CNN-MLPA.

2.1 Processing Element

Figure 1b shows the block diagram of a PE. Each PE has two input BRAMs (one for inputs and the other for weights), an output BRAM, control signals (Start,



(a) CNN-MLPA Architecture with PEs.

(b) PE structure containing AR and AF unit.

Fig. 1. Overview of the CNN-MLP accelerator.

and Done), an adder/multiplier unit (ALU), a controller, accumulation registers (AR), and an AF unit. AR is mainly an output buffer that stores the complete or partial result of a multiplication-accumulation (MAC) operation. Then this result is sent either back to the input buffer so that it can be used for the next layer or to the output stream interface. Each PE gives output for one neuron of a layer in case of MLP operation. For CNN, each PE may be used several times depending on the number of PEs, channels and filters in a convolution layer. Two types of AFs are implemented: a step function and log sigmoid. Before synthesis, two options are provided to the system designer to implement the AFs as either computation-based functions (synthesized hardware) or using LUTs.

2.2 Scheduler

The responsibility of the scheduler is to partition each layer of neurons into the linear array of PEs. The scheduler divides each layer into groups of neurons equal in size of the available number of PEs. If the number of neurons in a layer is not divisible evenly by the number of PEs, the remaining neuron(s) will be assigned to the first PE(s) during the next cycle. For example as shown in Fig. 2, with 4 PEs and 10 neurons in the first hidden layer, the scheduler will sequence two groups (G1, G2) of 4 neurons and one group (G3) of 2 neurons. The second hidden layer has 7 neurons. Thus, one group (G4) of 4 neurons and one group (G5) of 3 neurons will be scheduled. All neurons in a group are processed concurrently, while different groups are processed sequentially. Based on the scheduler assignment, the controller aligns weights and inputs for each neuron in each PE's internal BRAMs. The outputs of each group within a layer are saved in the output buffer, and assigned as inputs to the next layer for

MLPNN. In case of CNN, the outputs can be used as partials sum for the next channel in a convolution layer or as inputs for the next convolution layer.

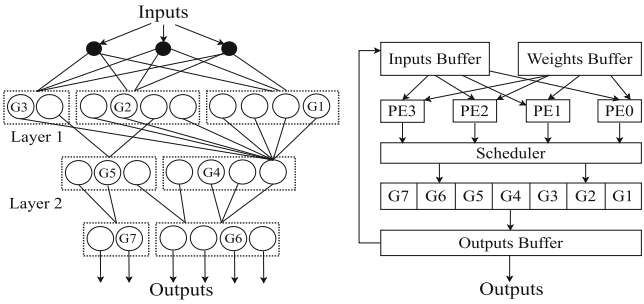


Fig. 2. Scheduling neurons in MLP with four PEs.

2.3 Controller

The controller organizes weights and inputs for the neurons. It divides the weight vector into groups, each having weights equal to the number of PEs, and allocates them into each PE's weight BRAM. It also connects the outputs of each layer with the appropriate weights, and this combination of outputs and weights is used in the next layer. The inputs are read serially from BRAM and stored in PE's input BRAM according to the number of neurons and PEs in the input layer for FCs. However, the allocation of inputs for CL depends on the input size, filter size, number of PEs, and strides. The allocation of weights and inputs for convolution operation in the convolution layer is briefly described by Fig. 3 for inputs with two channels, two filters and two PEs. Two sets of inputs in a channel can be convoluted by a filter in two PEs in parallel. Inputs (a, b, c) are arranged in PE-1's input BRAM. PE-2's input BRAM holds one stride size shifted version of inputs (b, c, d). Filter-1 has two sets of weights for Channel-1 and Channel-2, which are arranged in the temporary filter buffer. PE performs the MAC operations to produce partial sums. The same PEs are reused until convolution on Channel-1 is done. The same convolution process with Filter-1 is done on Channel-2. The outputs of both channels are accumulated in the output buffer. These operations are repeated for the same input channels with Filter-2 to produce output Channel-2. The output channels are used as input channels for the next layer.

The controller can read the AF values via the streaming input channel and store them in PE's BRAM if they are implemented as LUTs. Moreover, it calculates the number of weights being streamed to the CNN-MLPA based on the configured registers. Finally it streams the results out of the CNN-MLPA core and generates "done" signal in the output layer. It also enables users to switch between CNN and MLPNN operations.

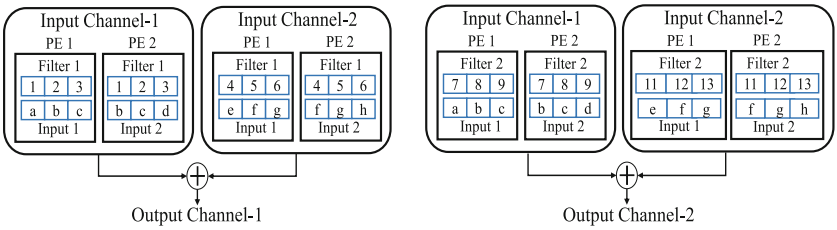


Fig. 3. Convolution layer operations.

2.4 Configuration Registers

The CNN-MLPA contains two sets of registers, one for the MLP block and one for the Convolution Layers (CL) block. They are used to specify the topology of the neural network during run time. The registers are described below with their corresponding parameters they store.

- Layers: number of layers in the FC block.
- Inputs: number of inputs.
- Outputs: number of outputs.
- Neurons$_{H_n}$: number of neurons in n^{th} hidden layer.
- ConvLayers: number of CL in the network.
- InputSize: input size in each CL.
- OutputSize: output size in each CL.
- Filters: number of filters in each CL.
- FilterSize: filter size in each CL.
- Channels: number of channels in each CL.
- Strides: the step size of the scanning filter in each CL.

The number of registers for MLP block scales according to the maximum network configuration such as: Maximum Number of Layers (MNL), Maximum Number of Neurons (MNN), Maximum Number of Neurons in Largest Layer (MNNLL), Maximum Number of Inputs (MNI), Maximum Number of Outputs (MNO), and the registers for CL scales according to the maximum number of filters, input and output channels, size of the filters and strides in the CL. Some values for the registers such as output size from each CL can be pre-calculated and then be sent to the accelerator if the network architecture is known. For example, output size is calculated by the equation, $OutputSize = \frac{\text{Input size-Filter size}+2\times\text{Padding Layers}}{strides}$. We can stream the value to the controller directly or let the accelerator calculate it.

3 Evaluation and Results for MLP Operations

3.1 Test Platform

The CNN-MLP accelerator is implemented on Xilinx Virtex-7 (xc7vx-485tffg1761-2) FPGA board. The overall implementation contains a softcore IP named MicroBlaze running at 100 MHz frequency as the processing system (PS)

and the CNN-MLPA as programmable logic (PL). The code for the accelerator was written in C++ and generated using Xilinx's Vivado-HLS 19.2 tool. After synthesis, we run C/RTL co-simulation with our testbench code in HLS to verify the functionality and output. Then, we export the RTL to vivado design suite where it is integrated with the MicroBlaze. With the help of a direct memory access (DMA) controller, communication among MicroBlaze, accelerator, and external memory (DDR3 DRAM) is established for transfer and storage of data. MicroBlaze provides programming interface to the users and enables communication with the accelerator via JTAG-UART. It transfers the input data and weights as a vector to the DRAM, and activates the DMA controller to transfer the data from DRAM to the local BRAM of the accelerator. It also assists the accelerator to be activated, read from and write to the storage. The block diagram of the overall architecture is shown in Fig. 4. AXI Timer IP is used to measure the time.

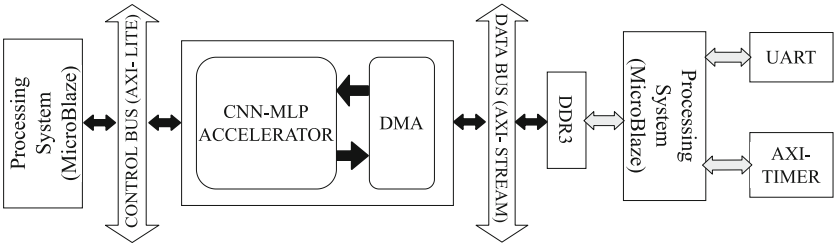


Fig. 4. Overview of the complete implementation.

3.2 CNN-MLPA Configurations

CNN-MLPA can be configured either for MLP operations or CNN operations. The MLP block will exist in both cases since CNN uses the FC layers of the MLP block. Therefore, when synthesized for CNN, it can perform MLP operations too. First, the CNN-MLPA was configured as an MLP accelerator only where the parameters are MNL, MNN, MNNLL, MNI, MNO as mentioned earlier. They are set as 6, 784, 784, 784 and 64 respectively before synthesis as a maximum bound for running the largest MLPNN we used as benchmark for MNIST dataset. Three versions of the accelerator with 4, 8 and 16 PEs and computation-based sigmoid AFs for all layers were synthesized. Then, we measured execution time and resource utilization. The results are compared with other FPGA implementations. Then we substituted sigmoid AFs with ReLU AFs, and floating-point precision with fixed-point for improving performance and resource utilization. Performance results for these tests are presented in Sects. 3.4 and 3.5.

3.3 Test Applications

Table 1 lists some referenced benchmarks along with the implementation platform, operating frequency and execution time. We ran these topologies on our architecture to evaluate performance of the MLP feature of the CNN-MLPA. Each application was first trained offline on a desktop PC using python. Different topologies were evaluated during the training phase. The validity of the results for each system was checked by comparing the outputs with the expected outputs produced by python, and C/RTL co-simulation result of HLS.

Table 1. Various FPGA implementations of MLPNN.

Works	Dataset	Topologies (Input, Hidden layers, Output)	Implementation platform	Frequency (MHz)	Execution time/Speed up
[17]	MNIST	#1: (784, 64, 64, 10)	Zynq Zedboard	200	36.1x Speed Up compared to 2.3 GHz Intel Core2 Processor
		#2: (784, 128, 128, 10)			
		#3: (784, 256, 256, 10)			
[18]		#4: (784, 600, 600, 10)	Virtex-7 (xc7vx485tffg1761-2)	490	2.514 µs
[19]		#5: (784, 1024, 10)	Zynq 7000	300	4.76 µs
[20]		#6: (784, 126, 126, 126, 10)	Zynq 7000	219	69 µs
[14]	IRIS	#7: (4, 7, 12, 3)	Kintex-7 (xc7k410t)	330	430 ns
[18]		#8: (4, 10, 3)	Virtex-7 (xc7vx485tffg1761-2)	490	79 ns
[21]	HAR	#9: (14, 19, 19, 7)	Spartan-6 (xc6slx45csg324-2)	67	800 ns

3.4 Performance Evaluation of MLP Accelerator

The benchmarks mentioned in Table 1 are run with our accelerator at 100 MHz for three different numbers of PE. Table 2 shows the execution time for 9 different topologies for 3 different numbers of PEs for floating-point precision. The throughput here is floating-point operations per second (FLOPS), which was calculated by the ratio of floating-point multiply-accumulate (MAC) operations and execution time. All the topologies were run by changing some parameters such as the number of inputs, outputs, layers, and number of neurons in each layer during run time without the need to re-synthesize. Input data sets were pre-loaded into off-chip DRAM. The execution time obtained from AXI timer of the FPGA includes the total time taken to setup the configuration registers with parameters, transfer input and weight data from DRAM to the accelerator's local BRAMs, read that data from BRAMs, store them into PE's internal BRAMs, compute results, write final results back into the BRAMs, and send the output to the DRAM. If AFs are used as LUTs, then they must also be loaded into the BRAM. This will increase the execution time. Therefore, the time increases as the network grows.

Table 2. Execution time of optimized MLP accelerator for different benchmark topologies.

Model	Topology (Inputs, Hidden layers, Outputs)	Execution time (μS)			Throughput for 8 PEs (MFLOP/**S**)
		4 PEs	8 PEs	16 PEs	
#1	784, 64, 64, 10	750	520	341	211
#2	784, 128, 128, 10	1,584	1,095	708	215
#3	784, 256, 256, 10	3,581	2,468	1,600	218
#4	784, 600, 600, 10	11,078	7,700	4,927	217
#5	784, 1024, 10	10,770	7,461	4,760	218
#6	784, 126, 126, 126, 10	1,950	1,244	798	212
#7	4, 7, 12, 3	6.1	6.00	5.88	49
#8	4, 10, 3	4.18	4.19	4.54	33
#9	14, 19, 19, 7	15.68	12.5	11	121

The computation time of CNN-MLPA accelerator is mainly dependent on the number of PEs and the size of the network. As the size of the two hidden layers increased from Model 1 to Model 4 of Table 2, we can see that the execution time increases. This trend will be different if convolution layers are used. Many hyper parameters of CNN affect its inference time. Most of the time inside PEs is spent on MAC operations. Moreover, computation-based sigmoid activation function contains exponential and division operations. These operations take many clock cycles. Now, if the network size grows for limited PEs, the number of groups of PEs will be high according to the partitioning technique described earlier in Sect. 2.2. The maximum number of PEs depends on the available resources of the FPGA platform. One PE can do several MAC operations at the same time. Applying loop unrolling pragma in Vivado HLS would process multiple loops in parallel, affecting the performance and resource utilization. This is one method for optimization in HLS based design. Here, the PEs are partially optimized with pipeline and unroll directives. All the loops are pipelined with initiation intervals that do not violate the timing constraints. Moreover, log-sigmoid activation function is replaced by ReLU activation function [22], which is very simple to implement on hardware and takes less clock cycles to execute. The execution time was brought down to half by this approach. If the number of PE is increased, more operations are executed in parallel, thus decreasing the time. However, for smaller network like the Models 7 and 8 of Table 2, the impact of large number of PEs is not significant because they will remain unused. The execution time can also be decreased by designing the accelerator to use LUTs for AFs and represent the data with fixed-point precision before synthesis. But it may reduce accuracy. We chose the bit width in such a way that the accuracy was preserved.

The results in Table 3 are derived for 16 PEs for 8 bit (4 bit integer part & 4 bit fractional part) inputs and weights at the input layer. Intermediate layers required at most 12 bits (8 bit integer part & 4 bit fractional part)

Table 3. Result comparison of floating-point and fixed-point precision for 16-PE design.

Model	Topology	Execution time (μS)		Accuracy at 8 Bit precision	Frame per second at 8 Bit precision	Throughput (OP/S) at 8 Bit precision
		32 Bit float	8 Bit fixed			
#4	784, 600, 600, 10	7,700	2,637	100%	380	634 MOPS
#3	784, 256, 256, 10	2,468	854	100%	1170	628 MOPS
#1	784, 64, 64, 10	520	180	99%	5556	610 MOPS

to maintain good accuracy. The precision was determined by analyzing the maximum and minimum values of inputs and trained weights with python script so that all the values within this range can be represented by fixed-point precision with minimum error. The precision for intermediate layers was determined by experiments because different network size may require different precision. Same accuracy as 32 bit floating-point precision was achieved in this method. Both the area and execution time were also reduced. Table 3 reports throughput in terms of both OP/S and frame per second (FPS).

3.5 Resource Utilization and Performance Comparison with Other Works

This section shows the resource utilization and performance of the programmable CNN-MLPA with MLP feature only. It also reports comparison on resource utilization and throughput with other MLP related works. The term normalized throughput (ratio of OP/S & total LUT or DSP utilized) was introduced for better comparison because different works adopt different parallelism strategies and use different FPGA platforms. Moreover, our design was not fully optimized for maximum resource consumption. We report the versions synthesized for running the largest network for MNIST dataset in Models 4 and 6 of Table 2. Model 4 represents the maximum number (600) of neurons in a layer and Model 6 represents the total number of layers (5), which are set before synthesis so that both can be run on the CNN-MLPA. The maximum number of 32-bit weights was chosen to be 850,000 because Model 4 requires around 835,000 weights. It almost exceeds the available BRAM resources. The input BRAMs of PEs also put pressure on on-chip memory. By directing Vivado HLS to allocate distributed RAM [23], also known as LUT-based RAM for PEs, the consumption of 36K

Table 4. Result comparison with other works.

MLP Designs	FPGA	LUT	DSP	BRAM	FF	Throughput (OP/S)	Normalized throughput [OP/($S \times LUT \times 1000$)]
CNN-MLPA (Our Work)	XC7VX485T	18,218	6	222	11670	610 M	33.5
NAFOSTED'17 [20]	XC5VLX-110T	218,528	–	–	139,391	3.8 G	17.4
FPL'21 [3]	XC7Z020	11,845	184	61	16,461	3 M	0.250
DLAU'17 [17]	XC7Z020	53,200	220	280	106,400	192 M	3.594
IJEECS'19 [2]	Altera 5CSEMA5F31C6	7,137	70	–	11,053	12.7 M	1.7
Xilinx FINN'17 [8]	ZC706	91,131	–	4.5	–	1.9 T	20849

BRAMs was brought down to 81.07%. Our design can also fetch additional weights from DRAM when required to avoid over utilization of BRAMs.

The CNN-MLPA shows better normalized throughput compared with all other works except Xilinx FINN in Table 4. Xilinx FINN outperforms us by a large number because it relies mainly on binary neural network training before inference. Thus, different networks need to go through the training cycle first. Furthermore, it can only be used with PYNQ boards [24] for interfacing with python, and it is not run time programmable. Our design is synthesized only once for all networks. Therefore, the latency increases with larger networks because the same resources are being utilized sequentially. Larger network will require more resources for equivalent performance. If the loops for the PEs in HLS can be unrolled efficiently, the resource utilization will be increased for better throughput.

4 Accelerator with CNN Feature

When CNN is run on the CNN-MLPA, both convolution layer (CL) block and MLP block shown in Fig. 1a are functional. Thus, the maximum limit for the configuration parameters of the convolution layers as described in Sect. 2.4 are also set based on the largest CNN being run. The largest CNN we ran was VGG-16 based on which the maximum values for input size, output size, number of filters and channels, filter size were chosen. Model specific parameters are sent during run time to execute different CNN topologies within the limit. We tested the programmable feature of the accelerator with three custom CNNs. Their network topologies and performance are reported in Table 5. These three CNNs perform MNIST digit classification. Some other benchmarks such as VGG-16, LeNet and SqueezeNet were also executed.

We used ReLU AF after each convolution layer. For the custom CNNs in Table 5, the whole convolution block is followed by two FC layers in the MLP block before outputs are generated. Thus, a CNN with one convolution layer was represented as 'Input→ Conv1→ ReLU→ FC→ ReLU→ FC→ ReLU→ Output'. The convolution layers have input and output channels. The number of output channels from a CL, which works as input for the next CL, depends on the number of filters used to scan the data of the input channels. The filters scan with a step size known as stride. Their weights are multiplied with the inputs and then the partial sums are accumulated. This operation is done in one PE. Thus, the same PE will operate several times. The number of times the same PE is used depends on the total number of PEs and input channels, and the size of filters and stride. An adder is used outside PE to sum up all the output of the same location of the channels (as shown in Fig. 3).

4.1 Results for Full CNN-MLP Acceleration

This section shows the performance and resource utilization when both convolution layer block and MLP block are operational. Table 5 includes result of

the execution time, frame per second, throughput and accuracy at fixed-point precision for three custom CNNs. 10 bit (6 bit integer part & 4 bit fractional part) precision for inputs and weights and 16 bit (6 bit integer part & 8 bit fractional part) precision for outputs and intermediate values were found to preserve the same accuracy as floating-point after some experiments on the 3 CNNs. The precision might be different for other very deep CNNs. It also shows the combinations of various parameters (filter, channel, stride, padding) used in the convolution layers. The number of MAC operations in the convolution layers is higher than the FC layers of CNN. The time also grows with the increase in convolution layers. The time can be reduced by using more PEs in CLs because they have more parallel operations than FC layers.

Table 5. Performance of CNN-MLPA for 3 CNN architectures.

Model	CNN architecture	Filter size (No.× Width × Height)	Input channels	Stride, padding	Test accuracy (%)	Execution time (mS)	Frame per second	Throughput (OP/S)
1	Input → Conv1 → ReLU → FC → ReLU → FC→ ReLU→ Output	Layer-1: (1 × 8 × 8)	Layer-1: 1	Layer-1: (1,0)	97	0.209	4784	0.65 G
2	Input → Conv1 → ReLU → Conv2 →ReLU → →FC →ReLU → FC→ReLU→ Output	Layer-1: (1 × 8 × 8)	Layer-1: 1	Layer-1: (1,0)	98	0.219	4566	0.92 G
		Layer-2: (3 × 3 × 3)	Layer-2: 1	Layer-2: (2,0)				
3	Input → Conv1 → ReLU → Conv2 →ReLU → → Conv3 →ReLU → →FC →ReLU → FC→ReLU→ Output	Layer-1: (3 × 3 × 3)	Layer-1: 1	Layer-1: (1,0)	99	0.237	4219	1.3 G
		Layer-2: (8 × 6 × 6)	Layer-2: 3	Layer-2: (2,0)				
		Layer-3: (3 × 3 × 3)	Layer-3: 8	Layer-3: (1,0)				

The comparison of resource utilization, throughput in terms of both OP/S and FPS and normalized throughput (ratio of OP/S & total LUT or DSP utilized) between our work and others for different benchmarks is shown in Table 6. It also contains some custom CNN implementation done by us and others using Xilinx DPU [25] on Zynq UltraScale+ MPSoC ZCU104 Evaluation board. It shows how CNN-MLPA can support various CNN networks using the same resources. The CNN-MLPA was not fully optimized for any particular network, but was optimized for all networks. Therefore, the normalized throughput is not the best but close to other custom designs, which supports only one network. The DSP utilization of CNN-MLPA is also lower than other designs. When compared with the Xilinx DPU, we got higher throughput of the models in Table 5 with CNN-MLPA when DPU was configured with single core. Our design is also portable to any FPGA while DPU is only supported by a few platforms.

Table 6. Result comparison with other works for different benchmarks.

Models	Designs	FPGA	LUT	DSP	BRAM 36k	FF	Throughput (OP/S)	Normalized Throughput [$\mathbf{OP}/(S \times LUT \times 1000)$]
LeNet-5	CNN-MLPA (This Work)	XC7VX485T	70878	96	361	58422	120 M	1.7
	Electronics'21 [26]	XCZU9EG	61,713	123	102	27,863	141 M	2.28
	ICEIC'20 [27]	XCZU9EG	32,598	143	95	33,585	201 M	6.14
VGG-16	CNN-MLPA (This Work)	XC7VX485T	70878	96	361	58422	29 G	418
	YUAN et al.'21 [28]	VCU118	781,000	4096	1779	243,802	2558 G	3275
	FCCM'21 [29]	XCVU9P	469,288	2100	27	663,488	49.92 G	106
SqueezeNet/ZynqNet	CNN-MLPA (This Work)	XC7VX485T	70878	96	361	58422	1.4 G	19
	Micro-processors and Microsystems'20 [30]	XC7Z020	38,038	172	97.5	25,036	5.5 G	145
	ARC'18 [31]	ZC702	13,418	149	124	18,114	1.1 G	87
Custom CNNs	Xilinx DPU (Single Core)-In Our Lab	ZCU104	49,383	710	255	98735	118 M	2.38
	Xilinx DPU (Dual Core) - Electronics'22 [25]	ZCU104	103,700	1,380	290	198,900	7 G	66
	Xilinx DPU - SEEDA-CECNSM'21 [32]	XC7Z020	31,812	194	117.5	58,169	4.1 M	0.128

5 Conclusion

In this paper, we presented a run time programmable accelerator on FPGAs to run both Convolutional Neural Network (CNN) and Multilayer Perceptron Neural Network (MLPNN) of any topology without re-synthesizing the accelerator every time for different networks. It partitions the operations of a network into groups of available processing elements (PEs). The advantages of this design are reusability and scalability over custom accelerators that can execute only specific DNN applications. The execution time and resource utilization are reported for some benchmark datasets to show how they vary with the number of PEs, precisions and activation functions (AF). It can be synthesized either for MLPNN or CNN. If synthesized for CNN, it can run both MLP and CNN applications. The synthesis is done only once after configuring parameters such as data precision, number of PEs, and implementation method for the AFs for a particular FPGA. Then it becomes efficient for handling a wide range of CNN and MLPNN topologies with varying accuracies and performance. Performance in terms of execution time may degrade for some networks, which can be considered as a trade-off for the flexibility, scalability and portability of the CNN-MLPA architecture.

References

1. Zhao, M., Hu, C., Wei, F., Wang, K., Wang, C., Jiang, Y.: Real-time underwater image recognition with FPGA embedded system for convolutional neural network. Sensors **19**(2), 350 (2019)
2. Ann, L.Y., Ehkan, P., Mashor, M.Y., Sharun, S.M.: FPGA-based architecture of hybrid multilayered perceptron neural network. Indonesian J. Electr. Eng. Comput. Sci. **14**(2), 949–956 (2019)
3. Ngo, D.M., Temko, A., Murphy, C.C., Popovici, E. FPGA hardware acceleration framework for anomaly-based intrusion detection system in IoT. In 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), pp. 69–75 (2021)
4. Jiang, W., et al.: Wearable on-device deep learning system for hand gesture recognition based on FPGA accelerator. Math. Biosc. Eng. **18**(1), 132–153 (2021)

5. Krips, M., Lammert, T., Kummert, A.: FPGA implementation of a neural network for a real-time hand tracking system. In: 2002 Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications, pp. 313–317 (2002)
6. Cheung, K., Schultz, S.R., Luk, W.: A large-scale spiking neural network accelerator for FPGA systems. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012. LNCS, vol. 7552, pp. 113–120. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33269-2_15
7. Liang, S., Yin, S., Liu, L., Luk, W., Wei, S.: FP-BNN: binarized neural network on FPGA. Neurocomputing **275**, 1072–1086 (2018)
8. Umuroglu, Y., et al.: FINN: a framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 65–74 (2017)
9. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
10. Wu, C.J., et al.: Machine learning at Facebook: understanding inference at the edge. In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 331–344. IEEE (2019)
11. Jouppi, N., Young, C., Patil, N., Patterson, D.: Motivation for and evaluation of the first tensor processing unit. IEEE Micro **38**(3), 10–19 (2018)
12. Aklah, Z., Andrews, D.: A flexible multilayer perceptron co-processor for FPGAs. In: Sano, K., Soudris, D., Hübner, M., Diniz, P.C. (eds.) ARC 2015. LNCS, vol. 9040, pp. 427–434. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16214-0_39
13. Majumder, K., Bondhugula, U.: A flexible FPGA accelerator for convolutional neural networks. arXiv preprint arXiv:1912.07284 (2019)
14. Sanaullah, A., Yang, C., Alexeev, Y., Yoshii, K., Herbordt, M.C.: Application aware tuning of reconfigurable multi-layer perceptron architectures. In: 2018 IEEE High Performance extreme Computing Conference (HPEC), pp. 1–9. IEEE (2018)
15. Fine, T.L.: Feedforward Neural Network Methodology. Springer, Cham (2006). https://doi.org/10.1007/b97705
16. Stanford University. cs231n convolutional neural network for visual recognition
17. Wang, C., Gong, L., Qi, Yu., Li, X., Xie, Y., Zhou, X.: DLAU: a scalable deep learning accelerator unit on FPGA. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **36**(3), 513–517 (2016)
18. Medus, L.D., Iakymchuk, T., Frances-Villora, J.V., Bataller-Mompeán, M., Rosado-Muñoz, A.: A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks. IEEE Access **7**, 76084–76103 (2019)
19. Abdelsalam, A.M., Boulet, F., Demers, G., Langlois, J.P., Cheriet, F.: An efficient FPGA-based overlay inference architecture for fully connected DNNs. In: 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–6. IEEE (2018)
20. Huynh, T.V.: Deep neural network accelerator based on FPGA. In: 2017 4th NAFOSTED Conference on Information and Computer Science, pp. 254–257 (2017)
21. Basterretxea, K., Echanobe, J., del Campo, I.: A wearable human activity recognition system on a chip. In: Proceedings of the 2014 Conference on Design and Architectures for Signal and Image Processing, pp. 1–8. IEEE (2014)
22. Si, J., Harris, S.L., Yfantis, E.: A dynamic ReLU on neural network. In: 2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS), pp. 1–6. IEEE (2018)

23. Fazakas, A., Neag, M., Festila, L.: Block RAM versus distributed RAM implementation of SVM classifier on FPGA. In: 2006 International Conference on Applied Electronics, pp. 43–46 (2006)
24. Python productivity for zynq. http://www.pynq.io/board.html
25. Hussein, A.S., Anwar, A., Fahmy, Y., Mostafa, H., Salama, K.N., Kafafy, M.: Implementation of a DPU-based intelligent thermal imaging hardware accelerator on FPGA. Electronics **11**(1), 105 (2022)
26. Cho, M., Kim, Y.: FPGA-based convolutional neural network accelerator with resource-optimized approximate multiply-accumulate unit. Electronics **10**(22), 2859 (2021)
27. Cho, M., Kim, Y.: Implementation of data-optimized FPGA-based accelerator for convolutional neural network. In: 2020 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1–2 (2020)
28. Yuan, T., Liu, W., Han, J., Lombardi, F.: High performance CNN accelerators based on hardware and algorithm co-optimization. IEEE Trans. Circ. Syst. I: Regul. Pap. **68**(1), 250–263 (2021)
29. Bhowmik, P., Pantho, J.H., Mbongue, J.M., Bobda, C.: ESCA: event-based split-CNN architecture with data-level parallelism on ultrascale+ FPGA. In: 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 176–180 (2021)
30. Mousouliotis, P.G., Petrou, L.P.: CNN-grinder: from algorithmic to high-level synthesis descriptions of CNNS for low-end-low-cost FPGA socs. Microprocess. Microsyst. **73**, 102990 (2020)
31. Mousouliotis, P.G., Petrou, L.P.: SqueezeJet: high-level synthesis accelerator design for deep convolutional neural networks. In: Voros, N., Huebner, M., Keramidas, G., Goehringer, D., Antonopoulos, C., Diniz, P.C. (eds.) ARC 2018. LNCS, vol. 10824, pp. 55–66. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78890-6_5
32. Flamis, G., Kalapothas, S., Kitsos, P.: Workflow on CNN utilization and inference in FPGA for embedded applications: 6th south-east Europe design automation, computer engineering, computer networks and social media conference (seeda-cecnsm 2021). In: 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), pp. 1–6 (2021)