IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 2, 2021, accepted October 27, 2021, date of publication November 9, 2021,
date of current version November 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3126838

Power Efficient Design of High-Performance Convolutional Neural Networks Hardware Accelerator on FPGA: A Case Study With GoogLeNet

AHMED J. ABD EL-MAKSOUD[1], MOHAMED EBBED[2], AHMED H. KHALIL[1], AND HASSAN MOSTAFA[1,3], (Senior Member, IEEE)

[1]Department of Electronics and Communications Engineering, Cairo University, Giza 12613, Egypt
[2]Department of Computer Engineering, Cairo University, Giza 12613, Egypt
[3]University of Science and Technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt

Corresponding author: Hassan Mostafa (hmostafa@staff.cu.edu.eg)

This work was supported in part by ONE Lab at Cairo University, and in part by the Zewail City of Science and Technology.

ABSTRACT Convolutional neural networks (CNNs) have dominated image recognition and object detection models in the last few years. They can achieve the highest accuracies with several applications such as automotive and biomedical applications. CNNs are usually implemented by using Graphical Processing Units (GPUs) or generic processors. Although the GPUs are capable of performing the complex computations needed by the CNNs, their power consumption is huge compared to generic processors. Moreover, current generic processors are unable to cope up with the growing CNNs demand for computation performance. Therefore, hardware accelerators are the best choice to provide the required computation performance needed by the CNNs as well as affordable power consumption. Several techniques are adopted in hardware accelerators such as pruning and quantization. In this paper, a low-power dedicated CNN hardware accelerator is proposed based on GoogLeNet CNN as a case study. Weights pruning and quantization are applied to reduce the memory size by 57.6×. Consequently, only FPGA on-chip memory is used for weights and activations storage without using offline DRAMs (Dynamic Random Access Memories). In addition, the proposed hardware accelerator utilizes zero DSP (Digital Signal Processing) units as all multiplications are replaced by shifting operations. The accelerator is developed based on a time-sharing/pipelined architecture, which processes the CNN model layer by layer. The architecture proposes a new data fetching mechanism that increases data reuse. Moreover, the proposed accelerator units are implemented in native RTL (Register Transfer Logic). The accelerator classifies 25.1 frames per second (fps) with 3.92W only, which is more power-efficient than other GoogLeNet implementations on FPGA in the literature. In addition, the proposed accelerator achieves an average classification efficiency of 91%, which is significantly higher than comparable architectures. Furthermore, this accelerator surpasses the popular CPUs such as Intel Core-i7 and GPUs such as GTX 1080Ti in terms of the number of frames processed per Watt.

INDEX TERMS Convolutional neural networks (CNNs), field programmable gate arrays (FPGAs), GoogLeNet, hardware accelerators, object classification, parallel computing.

I. INTRODUCTION

Deep learning has been employed in a lot of domains during the last decade, such as image classification [1], [2], object recognition [3], [5], object detection [6], [7], audio recognition [8], and self-driving cars [9], [10]. CNNs are used widely as they achieve challenging accuracies, and their models are easily applied to new applications.

CNNs are one of the common deep learning algorithms mainly used for image and video classification and

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

detection [11]. CNNs require large amounts of memory storage as there are millions of parameters in every CNN model. Moreover, CNNs are computationally intensive as they require billions of operations per image. The high computational complexity combined with inherent parallelism in these models makes them an excellent target for custom accelerators.

Although the CNNs have dominated the image classification and detection algorithms, there are two main challenges regarding their implementations [12]. The first challenge is the cost of computation, as their architecture consists of many convolutional layers, which are multiplication-hungry layers. The second challenge is the memory bandwidths, in which the memory fetching speeds are much lower than the processing speeds. These two challenges have raised the need to develop custom architectures to accelerate the CNN computations while keeping the power consumption at affordable rates for limited energy embedded applications. However, the variations of network architectures and data fetching patterns make it difficult to adopt one architecture for all CNNs. As a result, custom designs are the dominant approach for these networks to get the best performance across all performance metrics.

During the rising of deep learning (DL) and machine learning (ML) algorithms, two main categories of processors are used. The first platform is the Central Processing Units (CPUs), which are not efficient for DL and ML algorithms as these algorithms require high parallelism and a lot of DSP units to finish their processing rapidly. The second platform is the Graphical Procession Units (GPUs), which are capable of processing millions of pixels within a part of the second. Correspondingly, the GPUs are the most suitable platforms due to their high parallelism. Consequently, they have been used widely for both training and inference [13]. When it comes to hardware accelerators, FPGAs get a critical mission to provide high-performance – low power processing units [14], [15]. FPGAs stand for field-programmable gate arrays (FPGAs) that provide low power consumption, high parallelism, optimized hardware, and real-time computation capabilities. Moreover, FPGAs have the advantages of short time-to-market, reconfigurability, and reusable IP (Intellectual Property) options. There is another choice for designers, which is ASIC chips. ASIC is application-specific integrated circuits that provide the lowest power consumption and highest clock speeds, but it has a long time to market and high initial fabrication costs. These properties make it suitable for mass production, such as NVidia accelerators and Google Tensor Processing Units (TPUs) or data centers, such as Google cloud or Amazon Web Services (AWS).

As artificial intelligence (AI) is emerging increasingly in a lot of applications, the demand for hardware accelerators is increased. Recently, a lot of research is done to develop high-performance hardware accelerators for data centers, smartphones, and Internet of Things (IoT) devices. Accelerator specifications are set based on the target application, power consumption budget, and acceleration rate. AI accelerators need more specialized architectures and should be optimized for the target algorithm, in contrast to common architectures, such as RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) architectures. This approach is becoming more common in industrial and research applications, especially inference processors [16].

For many years, it is well-known that the depth of the network should be increased to get higher accuracies, especially the number of convolution layers. This has been a common direction till year 2014 when Szegedy proposed a new CNN network called GoogLeNet with the concept of inception module [17]. In this network, the depth and width of the network have been increased, but the computational budget has been kept constant by using the network-in-network concept. This concept uses additional 1×1 convolution layers to remove the network bottlenecks and to help in dimension reduction as shown in Fig. 1. GoogLeNet overcomes AlexNet [1] and VGG [2] networks by getting the highest accuracy with fewer weights. As AlexNet uses 60 Million weights to get 84.7% top-5 accuracy, and VGG-16 uses 138 Million weights to get 92.7% top-5 accuracy, GoogLeNet uses only 6.9 Million weights to get 93.4% top-5 accuracy. Despite all these advantages, GoogLeNet architecture is more complex than other CNN networks due to activations' data dependency and complex connections between inception layers. This makes it usually challenging for hardware accelerators designers except for few designs.

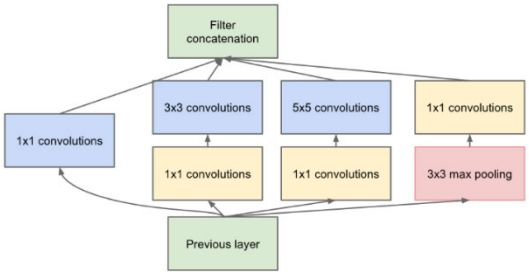

FIGURE 1. Inception module with dimension reduction [17].

The main features of the proposed accelerator in this work are highlighted as follows:

i. The accelerator achieves 25.1 fps for GoogLeNet classification with 3.92W only, which is more power-efficient than previous FPGA implementations for GoogLeNet CNN.
ii. It achieves an order of magnitude performance improvement over Intel Core-i7 and NVidia GTX 1080Ti.
iii. Weights pruning and quantization are used to cut down the memory usage by 57.6x. As a result, only FPGA Block RAMs BRAMs) are used for weights and activations storage without using offline DRAMs.
iv. It uses zero DSP units by converting all multiplications into shifting operations.

v. This accelerator is developed based on time-sharing/pipelined architecture that processes the CNN model layer by layer.
vi. It proposes a new data flow mechanism that leads to high data reuse and low power consumption.
vii. The processor uses simple eight distributed control units that can be reconfigured in the future to process other CNN models.
viii. The proposed accelerator uses only 224 simple parallel elements (PEs).
ix. The design achieves top-5 classification accuracy of 91%, which is significantly higher than comparable architectures.

This paper is organized as follows, Section II presents the related work. Section III presents the applied memory compression model using both weights pruning and quantization. Section IV explores the design options for the proposed architecture. Section V investigates the proposed architecture. Section VI shows the design discussions and experimental results. Finally, Section VII concludes the work.

II. RELATED WORK

Hardware accelerators design has become one of challenging topics in the research area. There are different implementations which are discussed briefly in this section. Firstly, Snowflake accelerator [18] which is able to achieve an average computational efficiency of 91%, and is implemented on a Xilinx Zynq XC7Z045. Snowflake is capable of achieving 128 Giga operations per second (GOPS/s) while consuming 9.48W of power. This work considers the number of frames without fully connected layers. Correspondingly, adding the fully connected layers overhead degrades its throughput and increases its power consumption. Moreover, it has high power consumption due to the usage of 1GB of DDR3 memory in addition to two ARM cores running at 800 MHz and one Kintex-7 FPGA. The entire system is clocked at 250MHz.

Another hardware accelerator that is designed by Zhao is synthesized by using the TSMC 65nm CMOS technology and achieves a peak of 280.8GOPS/s [19]. Its core area is 4.35mm^2 running at 650MHz with a power dissipation of 859mW. The input image/feature data and filter weight parameters are transferred from the external off-chip memory to the separated on-chip data buffer and parameter buffer. In addition, this work considers the number of frames without the implementation of the fully connected layers similar to [18], and correspondingly adding this overhead degrades its throughput and increases its power consumption.

DianNao accelerator is designed using CMOS 65nm technology with an area of 3.02mm^2 [20]. It performs 452GOP/s of fixed-point operations in parallel with 0.485W (excluding main memory accesses). This accelerator is 117.9× faster and 21.1× more energy-efficient than a 128-bit Single Instruction Multiple Data (SIMD) core. However, the reported throughput is the peak theoretical throughput only for some convolution layers without DRAM access time, which means that adding the DRAM access time overhead degrades the speed and increases the power consumption.

Eyeriss v2 is one of the popular hardware accelerators [21]. It is a fabricated chip with CMOS 65nm technology, which processes the convolution layers at 35fps for AlexNet at 278mW with 0.003mW for each DRAM access/multiply and accumulation (MAC). Furthermore, it performs 0.7fps for VGG-16 at 236mW. A network-on-chip (NoC) architecture is used for both multicast and point-to-point single-cycle data delivery to support the RS dataflow.

There are two popular high-level design flows of hardware accelerator implementation. The first flow is high-level synthesis (HLS), and the second one is Open Computing Language (OpenCL). They provide fast and easy hardware implementation, but they have a lack of optimization and energy efficiency. These high-level flows have been developed to build programs and execute them across heterogeneous platforms, such as CPUs, GPUs, and FPGAs [22]–[24]. HLS is an automated process to compile digital hardware circuits by synthesizing them. It enables building and verifying the hardware by giving better control over the architecture [25]–[27].

Moreover, many previous studies focus on accelerating the convolution layers of CNN only. For example, in [28] and [29], the hardware accelerator processes several convolution layers only rather than the full CNN while neglecting other CNN layers, such as fully connected layers. Consequently, those accelerators are not suitable to be deployed in low-power embedded applications.

III. MEMORY COMPRESSION

Increasing the size of the models has become a common trend in the development of CNN models. These models have a huge number of weights that require large memories. Moreover, 32-bit DRAM memory access requires 640pJ, as stated by [30], which leads to a fast battery drain of the embedded devices. Model compression techniques such as weights pruning and weights quantization are deployed in these CNNs models to reduce the memory size. Weights pruning is a process that removes unnecessary connections, which reduces the number of model weights. Removing these connections degrades the model accuracy as the model connections are mutually dependent. Correspondingly, retraining the remaining connections is a mandatory step to recover the accuracy loss [30].

Weights are usually represented with 32-bit precision, which allocates a large size of memory. Representing these weights in a smaller precision can compress the model significantly. This is the weights quantization operation that leads to smaller bit-representation [31]. After quantizing the model, many values of the weights are repeated frequently. Algorithms such as Huffman coding [32] are utilized to represent the most frequent values in smaller bits and the least frequent values in larger bits. Combining weights pruning,

weights quantization, and Huffman coding achieves larger compression ratios.

Applying memory compression on neural networks is an open area of research. Many techniques are proposed to deal with different models and get higher compression ratios. In [30], a pruning pipeline is proposed that firstly retrains the model from scratch, then performs the weights pruning iteratively, and retrains to compensate the accuracy loss due to the reduction of weights count. However, this model takes a large retraining time due to its iterative process. Moreover, the channels of the network can be pruned dynamically as stated in [33]. Finally, compression pipelines are proposed in [34], which consists of pruning, quantization, and Huffman coding to achieve a larger compression ratio.

A. MEMORY COMPRESSION MODEL

GoogLeNet model is compressed with a combined framework of weights pruning and quantization. The proposed framework consists of two stages which are selected carefully after exploring all related memory compression methods. Firstly, the framework applies the weights pruning based on dynamic network surgery work [35]. Secondly, weights quantization is applied based on incremental network quantization (INQ) framework [36]. The proposed framework is built without applying Huffman coding to avoid overhead latency of Huffman decoding while fetching the weights on the FPGA hardware. Fig. 2 shows a summarized flow chart for the used hyper-framework. Every framework for both weights pruning and weights quantization is discussed briefly in the following subsections.

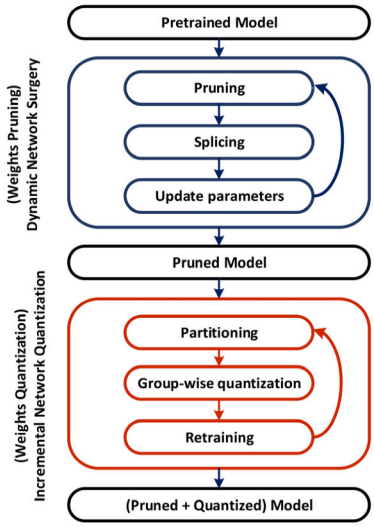

FIGURE 2. The proposed memory compression model.

B. WEIGHTS PRUNING

Weight pruning is performed using a dynamic network surgery method [35]. The pruning is performed on both, convolution layers and fully connected layers. Unlike the previous methods of alternating pruning and retraining, the dynamic network surgery method performs connections pruning and splicing for the network iteratively and implements the whole process dynamically. Firstly, activation masks are initialized for all weights to activate all of them. The masks are set during the process to one or zero to activate or deactivate them, respectively. During the forward propagation, the masks are element-wise multiplied by the weights, and the resulting outputs are used in the network. During splicing, the values of the masks change according to weights mean, and standard deviation. As a result, they might be reactivated for some weights to recover the connections that are found to be important during retraining. This results in making accuracy degradation insignificant.

C. WEIGHTS QUANTIZATION

After applying weights pruning model, weights quantization is used to shrink the precision from 32-bit to 4-bit. After analyzing multiple quantization frameworks, Incremental Network quantization (INQ) framework is used [36]. INQ is a group-wise quantization that is performed by partitioning the weights into two groups iteratively. Weights partitioning uses a pruning-inspired measure to split the two groups in each layer based on their values. The first group is quantized to the target precision, and the second group is retrained to compensate for the accuracy loss. Consequently, weights are iteratively quantized to 4-bit with a value of zero or a number with a power of 2's.

D. MEMORY COMPRESSION RESULTS

In this section, network training and memory compression results are presented, and some experiments on GoogLeNet CNN are demonstrated. GoogLeNet is built based on Szegedy's work [17]. The network structure is built as the reference paper, which is trained for 100 epochs on ImageNet Dataset. Furthermore, the optimization is done with stochastic gradient descent using a learning rate of 0.01, a momentum of 0.9, and a weight decay of 10^{-4}. Every 30 epochs, the learning rate is divided by 10. The network classifies images with 71.39% top-1 accuracy and 93.5% top-5 accuracy. In addition, the reference model has ∼6.9M weights with 32-bit precision.

Secondly, pruning is carried out with dynamic network surgery, which is performed on the convolutional layers, then fully connected layer as the full model pruning increases the accuracy loss. The model is pruned to have less than 1M parameters only to fit in Virtex-7 FPGA without using off-chip DRAMs. Consequently, the pruning is made aggressively to reach a 7.2× compression ratio with a top-5 accuracy loss of 1.4% and top-1 accuracy loss of 2.7% as listed in Table 1.

Finally, incremental network quantization is applied to the pruned model to reduce the precision of the weights to 4-bit instead of 32-bit. The removed connections are suppressed to zeros, and the quantization is performed iteratively on the remaining weights. At first, the accumulated partitions of

TABLE 1. Accuracy losses and compression ratios for different compression models.

Model	Error rate (%)			Compression ratio
	Top-1	Top-3	Top-5	
Reference model	0	0	0	1x
Pruned model	2.7	1.7	1.4	7.2x
Hyper model	4.8	3.5	2.6	57.6x

quantized weights at iterative steps are set as reference work as [0.2, 0.4, 0.6, 0.8, 1], but there is a sudden drop in the classification accuracy with 10% in top-1 accuracy. The sudden drop occurred because the model has many sparse weights. Therefore, the last steps starting from 80% quantization are increased to quantize the remaining weights. Consequently, the model is quantized using percentages of [0.2, 0.4, 0.6, 0.8, 0.85, 0.9, 0.95, 1], which yields a loss of 4.8% for top-1 error rate and 2.6% for top-5 error rate as listed in Table 1.

Quantizing from 32-bit to 4-bit leads to a compression ratio of 8× independently. Correspondingly, the hyper model of the weights pruning followed by quantization compresses the model with 57.6× successfully, as shown in the compression chart in Fig. 3. Furthermore, Fig. 4 shows the weights size reduction for each GoogLeNet layer for the plain model, pruned model, and quantized model with colors blue, gray, and orange respectively. It is observed from the chart how the pruning firstly reduces the number of parameters as shown with gray columns. Secondly, quantization makes a reduction with 8× for every layer which is shown with the orange columns. Moreover, the fully connected layer is the most compressed layer as it has many weights that tend to zero. On the other hand, 3×3 convolution layer come at the second most compressed layer as it has many filters per layer.

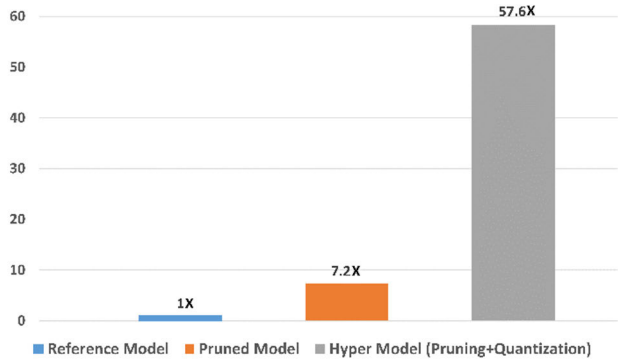

FIGURE 3. Compression ratios after pruning and quantization.

IV. DESIGN EXPLORATION

A lot of hardware accelerators propose high throughput on feed-forward CNN networks in the literature such as LeNet, AlexNet, and VGG [16], [37]. These hardware accelerators fail to process the inception network well, and the obtained speed is degraded because of the structure of the inception module. The inception module increases the depth of the layers horizontally and vertically while keeping computational cost by adding 1×1 convolution layers. Although this improves the accuracy, it increases system complexity.

The proposed processor is designed to fit GoogLeNet inception CNN. GoogLeNet CNN achieves higher inference accuracy while keeping the weights count of ∼6.9 Million only, which is a great improvement compared to previous CNN networks. Moreover, the number of weights is cut down significantly after performing weights pruning and quantization as investigated in the previous section to be able to load them on FPGA BRAMs without using offline DRAMs.

TABLE 2. Popular CNNS in literature.

CNN	Year	Top-5 accuracy (%)	Number of weights (Millions)
AlexNet	2012	83.6	60 M
VGG-16	2014	92.7	138 M
GoogLeNet	**2014**	**93.4**	**6.9 M**
Inception V3	2015	96.5	23.6 M
ResNet-50	2016	96.4	25.6 M

In Table 2, a summarized comparison is made between popular CNN networks. VGG-16 has a Top-5 accuracy with 92.7%, but it requires 138 Million weights per frame. This requires a huge memory size, which in turn will increase the computation load and memory access. On the other hand, Inception V3 and ResNet-50 get higher accuracy with 96.5% and 96.4%, respectively. They get an approximate accuracy improvement of 3% with 3.5× memory storage. Accordingly, it is clear that GoogLeNet achieves the best accuracy while keeping the number of weights in an acceptable count.

GoogLeNet has 57 convolution layers and only one fully connected (FC) layer. The computation workload is centered in convolution layers with 2.58G MACs. Furthermore, the fully connected layer uses a huge number of weights per layer with 1.024M weights. Moreover, it has fourteen Maxpooling layers to reduce the input feature map size. The network has one average pooling layer to reduce the input feature map size before the fully connected layer. Finally, Softmax layer is used to get the classification results in probabilistic values. Table 3 shows the detailed architecture and design parameters of GoogLeNet.

CNN processing is computation-intensive for both convolution layers and fully connected layer. Correspondingly, parallelism is required to reach a short inference time. There are different ways of parallelism in CNNs, such as batch parallelism, inter-layer parallelism, inter-feature map parallelism, inter-convolution parallelism, and intra-convolution parallelism, as stated by [14]. Every hardware accelerator adopts one or more of these parallelism types to get the target throughput. In the proposed accelerator, 24 kernels of 3×3 layers, nine kernels of 5×5 Convolution layers, or four kernels of 7×7 Convolution are processed in parallel as shown in Fig. 5 and stated in Table 4. The following parallelizing techniques are adopted:

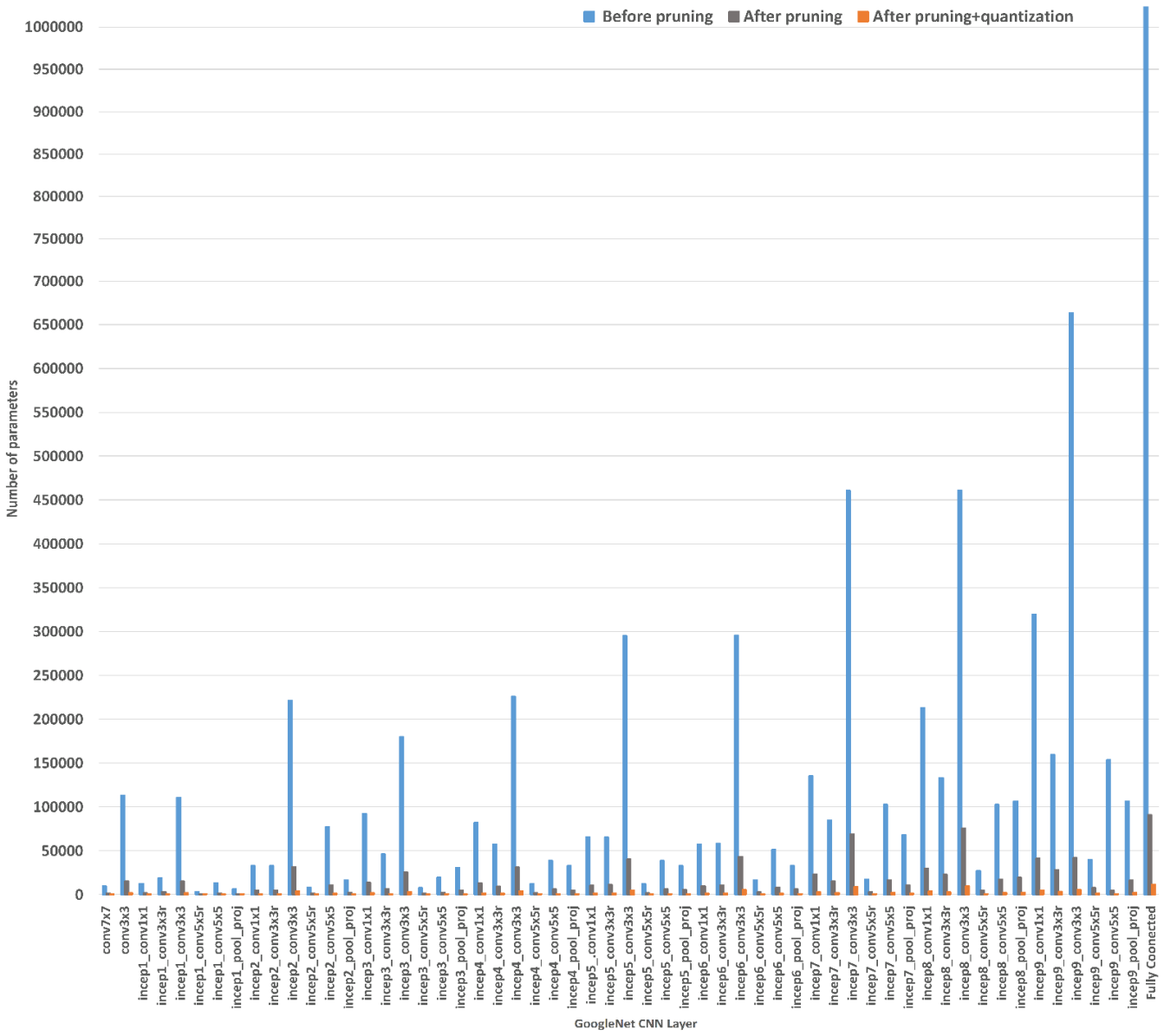

FIGURE 4. Weights compression results for GoogLeNet layers.

A. INTER-LAYER PARALLELISM

In inter-layer parallelism, the hardware accelerator has a feed-forward hierarchical structure that processes a succession of data-dependent layers. They are executed in a pipelined fashion by executing a layer while preparing the next layer to be processed. In this way, the hardware accelerator utilized area is decreased significantly, which makes it easy to fit it in FPGAs or develop a small chip.

B. INTRA-FEATURE MAP PARALLELISM

In intra-feature map parallelism, a group of the output feature map pixels of a single output feature map plane is processed in parallel, which reduces the required processing time by acceleration factor X. This type of parallelism depends on the output feature map and kernel sizes.

C. INTRA-CONVOLUTION PARALLELISM

The last adopted parallelism is the intra-convolution parallelism, in which the processing of 2D convolution layers is implemented in a pipelined/parallel fashion.

There is always a trade-off between the acceleration factor and the accelerator size during selecting the suitable number of PEs. Firstly, an analysis of GoogLeNet CNN layers is made to determine the suitable number of PEs. GoogLeNet has four different convolution kernel sizes, which are 1×1, 3×3, 5×5 and, 7×7 kernels. The convolution opcode is represented by two bits to select the convolution type in different

TABLE 3. GoogLeNet analysis.

GoogLeNet CNN	Count
Convolution layers	57
Convolution layers in depth	21
Convolution workload (MACs)	2.58G
Convolution parameters	5.9M
Activation layer	ReLU
Maxpooling layers	14
Average pooling layers	1
FC layers	1
FC workload (MACs)	1.024M
FC parameters	1.024M
Total workload (MACs)	2.58G
Total parameters	~6.9M

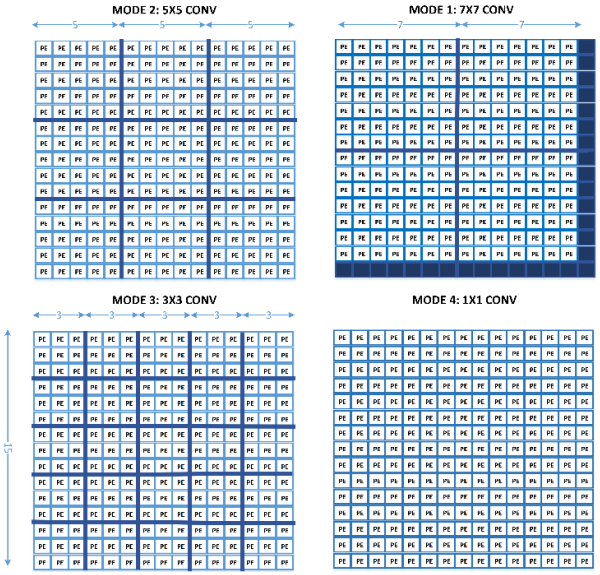

FIGURE 5. Different kernel sizes on PEs.

TABLE 4. Required #pe per kernel.

Kernel size	#PEs/kernel	Convolution opcode
7x7 Convolution	49	00
3x3 Convolution	9	01
5x5 Convolution	25	10
1x1 Convolution	1	11

blocks by the control unit as listed in Table 4. In addition, the number of PEs is chosen to be 224 PEs that processes 224 kernels of 1 × 1 Convolution kernels.

The capacity of memory in FPGAs is not large enough to save all weights and feature maps (FMs) of all CNN layers. Consequently, loop tiling is used to fetch the upcoming parts of feature maps in addition to kernel weights while processing the current layer. Feature maps and kernels of convolution layers are batched so that kernel weights are loaded only once, and FM tile is loaded once per batch. This factorization is employed to increase the data reuse and computational throughput. Fig. 6 shows an example of input feature map (IFMAP) tile to PEs fetching for 5 × 5 convolution. Convolution layer pseudo-code for one layer is shown in Fig. 7, which consists of nested for-loops. The first two for-loop iterate over the output feature map. The U for-loop iterates over the output channels. Also, the for-loops of V and iterates over input channels. Finally, the last two for-loops iterate over kernel rows and columns.

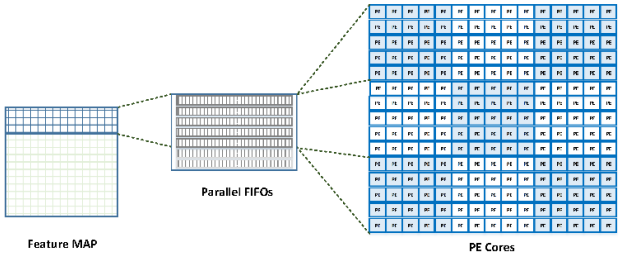

FIGURE 6. Reading IFMAP tile from main buffer to PEs – 5 × 5 convolution example.

```
for (n=0; n<N; n++) {                    //output FM rows
 for (m=0; m<M; m++) {                   //output FM columns
  for (u=0; u<U; u++) {                  //output channels
   for (v=0; v<V; v++) {                 //input channels
    for (i=0; i<K; i++) {                //kernel rows
     for (j =0; j<K; j++) {              //kernel columns
        Fout[u][n][m]+=Fin[v][S*n+i][S*m+j]*K[u][v][i][j]
   } } } } }
        Fout[u][n][m]+=bias[u]
}
```

FIGURE 7. Convolution layer pseudo-code.

Some loops are selected to be unrolled to speed up the processing and parallelize the processing of certain iterations on the hardware. The number of parallelized iterations is called the unroll factor. Selecting suitable unroll factors might lead to huge hardware utilization. For the proposed processor, the for-loops of rows and columns are completely unrolled. Moreover, the for-loops of feature map rows and columns are tiled with a size of feature map row. The tile is reused by shifting the rows up by the stride value and other rows are reused again. Finally, the output channel is parallelized by processing multiple kernels and writing out multiple output pixels in parallel.

V. ARCHITECTURE

The architecture is built as a time-sharing processor that performs the computations for CNN layers. Therefore, the processing flow is made depending on the accelerator's control units and CNN structure. The design of each unit is discussed in this section by showing its specs and implementation. Finally, several general modifications are adopted

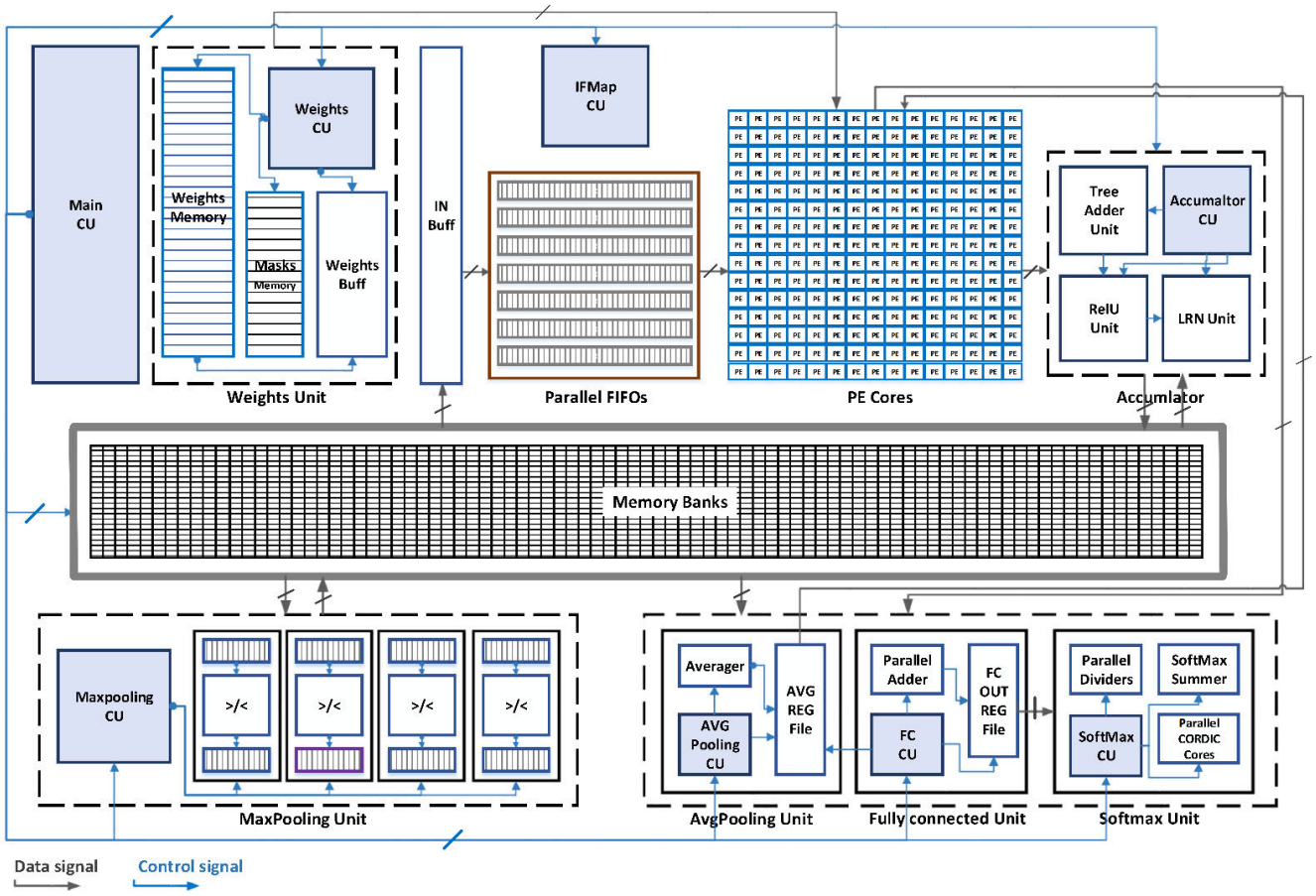

FIGURE 8. Top-level diagram of the proposed architecture.

to improve the proposed accelerator and make full use of observed enhancements after memory compression results. The proposed state-of-art hardware accelerator consists of 256 memory banks, 224 parallel elements, weights memory, accumulator unit, maxpooling unit, average pooling unit, fully connected unit, softmax unit, buffers, and nine distributed control units. Each unit is carefully designed and implemented in native RTL (Register Transfer Logic) to get the best performance and lowest power consumption. The top-level diagram of the proposed architecture is shown in Fig. 8. The design of these units is discussed in this section by showing the specs and improvements for each block.

A. MEMORY ORGANIZATION

Memory organization is one of the main challenges during accelerator design. As discussed earlier, the memory bottleneck requires careful handling and planning. The final memory organization is set after analyzing several options to select the best implementation. Firstly, the limited number of access ports of the intermediate memory is overcome by dividing the memory into 256 banks to read/write in parallel. Secondly, adding multiple buffers resolves the stalls. In the proposed accelerator, separate memories for weights and temporary data are used.

The proposed architecture consists of multiple hierarchy levels of storage as follows:

- It consists of 256 memory banks to save the partial summations during computations. They are implemented in FPGA BRAMs.
- Weights memory saves all weights of the CNN model. It utilizes 3Mb on FPGA BRAMs.
- Weights Masks memory saves all weight masks. If the weight is a non-zero value, its value is fetched from the weights memory.
- Weights buffer fetches the weights from weights memory and prepare them for parallel fetching.
- IFMAPs buffer loads the FMs from the memory banks and prepares them for FIFOs (First In First Out) blocks.
- Seven parallel FIFOs are used to load complete seven rows from the input buffer. They save it while convoluting them with filter kernels.
- The internal register in each PE saves the loaded weight till the PE finishes.

This mechanism results in high data reuse because it enables global fetching for all loaded kernels with the same

loaded feature map part on FIFOs. In addition, it empties the input buffer to load more IFMAPs. This mechanism is designed by considering the latency of buffer loading to illuminate any stalls during convolution. The next row of IFMAP is loaded while convoluting the loaded rows on FIFOs except for 7×7 convolution as it has a stride with two, which shifts out two rows every shifting up.

Weights memory saves the weights with 12-bit word length. After quantizing all weights, the weight's word length has become 4-bit, which makes the memory store $3\times$ more weights than before. The weights buffer prepares the weights for parallel shifting to the processing unit based on current convolution layer sizes. The weights fetching scenario goes as follows, the weight control unit (WCU) checks the next bit mask. If the bit is zero, it writes a zero in the weights buffer. If it is equal to 1, the WCU reads the weight value from the weights memory and writes it into the weights buffer. The design is verified against any stalls, so the next weights become ready while the processing unit is running the currently loaded weights.

B. PROCESSING UNIT

The processing unit (PU) consists of 224 parallel elements (PEs), summation unit, and PU control unit. The PE consists of one multiplier in addition to two multiplexers as shown in Fig. 9. The first multiplexer is for input weight that selects between the stored weight or a new one. The second multiplexer selects the desired input activation based on convolution kernel sizes, such as 1×1, 3×3, 5×5, and 7×7 Convolutions or FC inputs. The multiplier is built with a simple shift right block as all input weights are quantized to multiple of 2's number. The summation unit is built of hierarchal adders to reduce the number of adders for different convolution sizes. This is resolved by using 24 adders with nine inputs only instead of many adders with different input sizes.

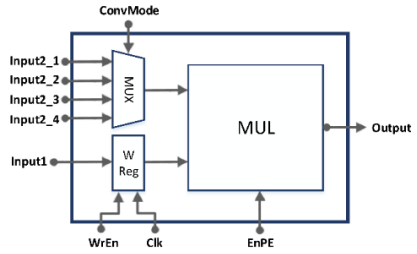

FIGURE 9. Parallel Element structure.

C. CONTROL UNITS

The proposed hardware accelerator runs based on eight distributed-hierarchal control units (CUs) in addition to the main CU to simplify the overall design. Every CU is controlled by the main CU. Moreover, every CU controls all its related signals. Each CU is designed with a finite state machine that runs based on the stored control words. This makes it easier to adapt and run other CNNs by changing the control words only. The CUs are as follows:

- Main CU
- Processing unit CU
- Partial sum accumulation CU
- IFMAP fetching CU
- Fully connected CU
- Maxpooling CU
- Averagepooling CU
- Softmax CU

D. FULLY CONNECTED UNIT

As discussed before, fully connected (FC) layers are memory-centric. They usually contain millions of weights, and each weight is used only once. As each weight in FC layers is used in one inference process only, it leaves no chance for data reuse. The limited bandwidth significantly degrades the performance as reading those weights takes a long time, so it requires a careful design for this unit.

Firstly, a fast analysis for FC layer is presented. (1a) and (1b) represent a pseudo-code for the FC layer. The output of average pooling is 1024 activations, which is the N value. It is stored in an intermediate buffer as input activations for FC, then it is fetched to the PU. The network is trained on the ImageNet dataset with 1000 classes, so the M is equal to 1000.

$$out_m = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} W_{mn} x A_n + B_m \tag{1a}$$

$$for(m = 0; \quad m < M; m++)\{$$

$$for(n = 0; \quad n < N; j++)\{$$

$$out_m += W_{mn} x A_n\}$$

$$out_m += B_m\} \tag{1b}$$

E. MEMORY MANAGEMENT

Fully connected processing requires 256 weights every cycle in the proposed design, which is not valid if they are fetched from weights memory directly. After performing memory compression, as discussed in section II, a lot of weights are suppressed to zero after weights pruning, especially in the fully connected layer. An analysis is made on fully connected weights to discover their weights values. It is found that the number of non-zero weights per 256-tile does not exceed 32 weights. This makes it easy to decompress 256 weights per cycle while knowing that there are 32 non-zero weights by maximum. The decompression unit is implemented and integrated with weights unit to use it while FC processing without any memory stalls.

Moreover, the input activations are fetched tile by tile with 256 tile size and used for 1000 cycles before fetching the next tile. The technique leads to high data reuse for activations instead of reading and writing them every cycle.

F. COMPUTATION MANAGEMENT

The PEs are used for FC multiplications with extra 32 shifting blocks to make full use of the processing unit. The acceleration of FC is made for the inner loop by processing a tile of 256 weights each cycle. Therefore, the inner loop is processed in 4 cycles instead of 1024 cycles. Consequently, the fully connected layer is accelerated by 256× more than a single MAC unit. The tilling diagram is shown in Fig. 10. The diagram illustrates the process of the adopted fully connected computations. The flow is as follows:

- New 256 weights are fetched to PU every cycle.
- An input activation tile is updated every 1000 cycles.
- The multiplications is performed and forwarded to the parallel adder every cycle. Finally, the output is saved to the output register file.
- After the first inner loop of pseudo-code in (1b) is completed, the output of every summation is added to its corresponding value in the output register file, and so on till finishing all tiles.

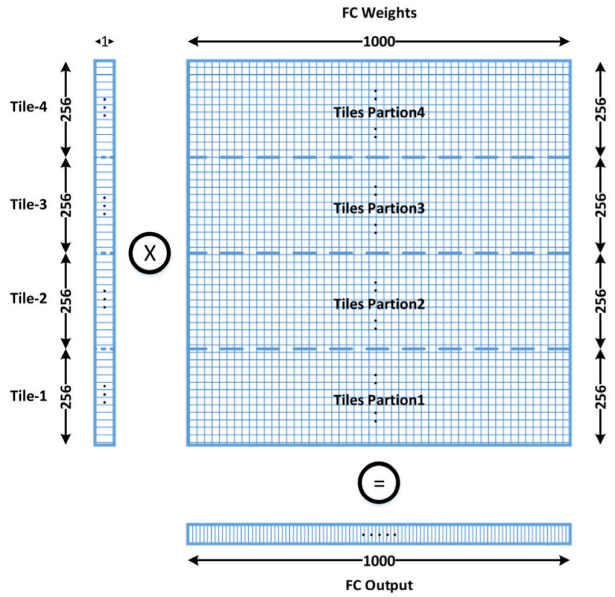

FIGURE 10. Fully connected layer tiling diagram.

G. MAXPOOLING UNIT

Maxpooling is used between convolution layers to reduce the spatial size of feature maps. There are 14 Maxpooling layers in GoogLeNet. Maxpooling unit works on four feature maps in parallel. As a result, the unit uses four maxpooling parallel blocks. Every block consists of an input buffer, output buffer, and comparators.

H. LOCAL RESPONSE NORMALIZATION UNIT

Local response normalization (LRN) is used to normalize the distribution of the input activations by normalizing over local input regions. It depends on the activations of adjacent kernels at the same layer [40]. This is made instead of computing mean and deviation as performed by the batch normalization (BN) layer.

LRN does not have any learnable parameters and all computations are made between input activations as shown in (2). The parameters (α, β, k, n) are set firstly $\gamma = 0.0001$, $k = 1$, $\beta = 0.75$, and $n = 5$. The parameter n represents the number of input activations $a^j_{x,y}$ that is squared and summed to compute the normalized activation. After investigating the LRN equation, it needs a lot of computations to generate normalized activations. Squaring, division, and powering blocks in addition to intermediate registers are needed, which takes up a large area and power to compute it.

$$b^i_{x,y} = \frac{a^i_{x,y}}{(k + \alpha \cdot \sum_{j=\max(0,i-\frac{n}{2})}^{\min(N-1,i+\frac{n}{2})} (a^j_{x,y})^2)^\beta} \tag{2}$$

Instead of building these large blocks, a software experiment is done on the Goog t testing set to get the average difference before and after the LRN layer. This average difference is computed across input channels and testing images. The average values are ranging from 0 to 0.006, which are added randomly across input channels instead of making all these computations. The overall accuracy does not decrease as it is well known that the CNNs themselves add up noise through different layers. This is proven experimentally by replacing LRN with a randomizer using patches of testing images, every patch contains 128 images. The overall accuracy is ranging from 0.02 to −0.02 or does not change in several testing patches. This is done by using random values from the previous software experiment.

I. AVERAGE POOLING UNIT

The average pooling layer is added before the fully connected layer to reduce the input feature map size to 1×1024 instead of $7 \times 7 \times 1024$. It simply adds up all pixels of every 7×7 feature map and divides it by 49. The unit works on eight feature maps in parallel and stores the output in an intermediate buffer for the fully connected layer.

J. SOFTMAX UNIT

Softmax unit is used to convert the output of a fully connected layer to probability distributed values [39]. The unit consists of 10 parallel CORDIC (COrdinate Rotation DIgital Computer) blocks to compute the exponential function. The unit stores exponential outputs again in the buffer while computing their summation. After computing the summation of 1000 exponentials, every exponential is divided by the summation and stored in the final output buffer. As shown in (3), N is equal to 1000 as same as the number of CNN classes. The block diagram is shown in Fig. 11.

$$f(i) = \sum_{i=0}^{N} \frac{e^{a_i}}{\sum_k e^{a_k}} \tag{3}$$

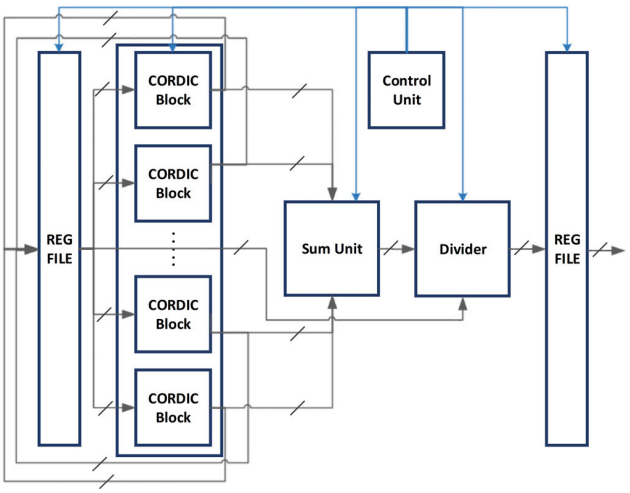

FIGURE 11. Softmax unit structure.

K. PROCESSOR MODIFICATIONS

The DSP resources of the FPGA are firstly used to implement PEs multipliers, which increase the power consumption while processing CNN layers. After memory compression and quantization, the weights are quantized to 4-bit only, and they become one of a few distinct values. As a result, the multiplication is made simply by shifting after decoding these weights based on the decoding table in Table 5. This modification lets the processor be DSP-free, and the power consumption of multipliers is saved as the multiplication has become a simple rewiring instead of large conventional adders.

TABLE 5. Weights decoding table.

Weight value	Decoded Code	Shifting	Sign
0.5	0001	>>1	+ve
0.25	0010	>>2	+ve
0.125	0011	>>3	+ve
0.0625	0100	>>4	+ve
0.03125	0101	>>5	+ve
0.015625	0110	>>6	+ve
0.0078125	0111	>>7	+ve
-0.5	1001	>>1	-ve
-0.25	1010	>>2	-ve
-0.125	1011	>>3	-ve
-0.0625	1100	>>4	-ve
-0.03125	1101	>>5	-ve
-0.015625	1110	>>6	-ve
-0.0078125	1111	>>7	-ve

Also, convolution kernels with equal size are processed at the same time which make some of the parallel cores are unutilized during layers computations. This is resolved by enabling the processing of multiple kernel sizes in parallel, which increases the utilization of the cores. Finally, the time overhead for writing and reading all padding pixels is skipped to save these cycles. Consequently, nearly 240,000 cycles are saved for writing and thousands of cycles for reading.

VI. DISCUSSION AND RESULTS

In this section, testing of the proposed design is discussed, and the experiment of selecting the fixed-point precision is presented. In addition, the theoretical throughput and the resource utilization of the proposed processor are reported. After that, a comparison is made between Intel Core-i7 CPU, NVidia GTX 1080Ti GPU, and the proposed accelerator to show the power consumption improvement. Finally, a comparison between the existing GoogLeNet implementations and the proposed accelerator is provided.

Design testing is an important step to validate the design functionality. Firstly, testing and validation for each independent unit are done by testing the unit with critical cases to resolve any issue. The testing for each unit is done interactively to trace every signal and try different inputs. The integration is performed gradually between the design units. After integrating all units, a top-level test bench is used to test and validate the proposed hardware accelerator. Testing images are converted earlier to binary RGB format and written in separate files. The test bench loads the image on the FPGA, and the processing is enabled by the "Start_CNN" signal. The softmax unit computes the highest class probability. Finally, the class number is mapped to its class name.

The effect of word length is tiny on the accuracy of convolutional neural networks, as stated in the literature [40], [41]. Arithmetic operators with 12-bit fixed-point are used instead of 32-bit to reduce storage size and power consumption during operations. Several experiments are held to select the suitable arithmetic operator width while keeping the accuracy loss at least value. The experiments are done on an epoch of 1024 images from the ImageNet dataset to see the effect of sweeping the word length. The model is implemented in software by providing the maximum and minimum values that are represented by the accelerator, and every output activation of every layer is suppressed to zero or truncated to this word length.

The first experiment is done to select the integer part width. Width of 4-bit is selected for the integer part to represent the maximum integer activation, which keeps the accuracy without any loss. The second experiment is done for the fractional part while fixing the integer part at 4-bit. Fig. 12 shows top-1, top-3, and top-5 error rates when using 16-bit to 9-bit fixed-points. The number of bits represents the whole word length. For example, at 14-bit word length, the fractional part is 10-bit. The usage of 8-bit word length gives the worst accuracy with a loss of nearly 30%. By increasing the length gradually, the error rate starts to decrease to zero at 15-16 bits. It is observed that using 12-bit width with an 8-bit fractional part gives an error rate of 0.01 while keeping it multiple of four. The word length reduction experiment is done on the inference phase only to use it by the accelerator, so while training, all values are represented by full precision. On the other hand, theoretical throughput is calculated to compare it

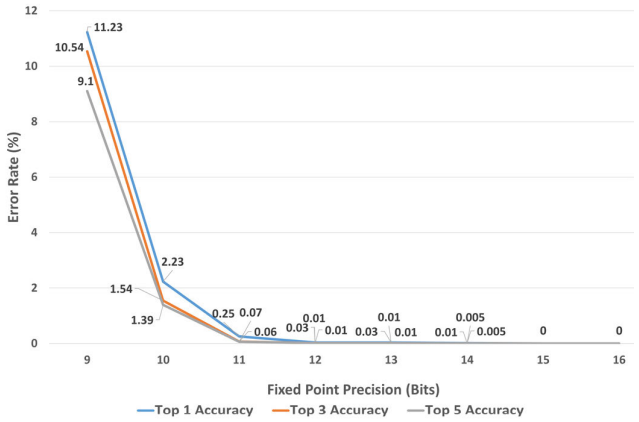

FIGURE 12. Error rate versus fixed point precision.

with the measured throughput. It is computed as follows:

$$\#\text{convolution cycles/frame} = \sum_{l=1}^{L} \frac{\text{kernel size}}{\#parallel\ kernels}\ x\ OFMAP\ Size\ x\ \#IFMaps \tag{4}$$

where the OFMAP stands for output feature map and IFMAP stands for input feature map. (4) calculates the needed number of cycles for all 57 GoogLeNet convolution layers. Also, this count is added to the needed cycles for maxpooling, average pooling, fully connected, and softmax layers processing. The theoretical throughput is 30.3fps at a frequency of 200MHz.

The proposed hardware accelerator is implemented in native RTL (Verilog) on Virtex-7 FPGA. The resources of Virtex-7 FPGA are suitable for the proposed implementation due to two reasons. Firstly, the on-chip BRAMs of Virtex-7 are 52,920Kb, which are used for weights and intermediate layers storage. Secondly, the number of LUTs is suitable to synthesize the design with 433,200 LUTs. Table 6 shows the system utilization on the FPGA. Thanks to memory compression, the accelerator is built without using DSP units, and the memory is synthesized using on-chip BRAMs only. By using Vivado power analyzer, the proposed hardware accelerator classifies 6.4 frames per Watt as shown in Table 7.

TABLE 6. The proposed accelerator's utilization on VC709.

Resource	DSP	BRAM	LUT	FF
Used	0	1134	407290	85927
Available	3600	1470	433200	866400
Utilization	0%	77%	94%	10%

The proposed accelerator runs at a maximum frequency of 200MHz. As shown in Table 7, a comparison is made between Intel Core-i7 CPU, NVidia GTX 1080Ti GPU, and the proposed accelerator in terms of the operating frequency, process technology, power consumption, performance (fps), and power efficiency (fps/W). The results show that the proposed accelerator provides the best performance

TABLE 7. Comparison with other platforms.

	Intel Core-i7	NVidia GTX 1080Ti	This work
Frequency	3.1GHz	1.5GHz	200MHz
Technology	22nm	16nm	28nm
Power (W)	15	106	3.92
Performance (fps/s)	1.916	85.83	25.1
Performance[(1)] (fps/s)	0.124	11.45	25.1
Power Efficiency (fps/W)	0.128	0.81	6.4

[(1)] Normalized performance to 200 MHz frequency

in terms of the number of frames per Watt. The power efficiency is 6.4 frames/Watt for the proposed accelerator, 0.81 frames/Watt for NVidia GPU, and 0.128 frames/Watt for Intel Core-i7. It is worth mentioning that the used FPGA is fabricated with 28nm technology, which dissipates more power than 14nm and 22nm technology nodes. It has 49.5× power consumption improvement over Intel core-i7 and 7.8× over NVidia GTX 1080Ti.

The developers have started to use embedded AI accelerators for deploying their deep learning applications. NVidia Jetson Nano and Intel Movidius NCS are from these popular accelerators. Table 8 shows the comparison between the proposed hardware accelerator, NVidia Jetson Nano, and Intel Movidius. Firstly, Jetson Nano is used to run GoogLeNet using two frameworks: Caffe and TensorRT at a frequency of 920MHz [44]. Caffe framework is widely used in deep learning development, while TensorRT framework is developed by NVidia to accelerate the inference process. Secondly, Intel Movidius NCS (Neural Compute Stick) runs GoogLeNet using

TABLE 8. Comparison with popular embedded AI accelerators.

	NVidia Jetson Nano [42]	NVidia Jetson Nano [42]	Intel Movidius NCS [43]	This work
Framework	Caffe	TensorRT	Caffe	-
Frequency	920MHz	920MHz	933MHz	200MHz
Power (W)	5	5	-	3.92
Performance (fps/s)	19	60	13.66	25.1
Performance[(1)] (fps/s)	4.13	13.1	2.93	25.1
Power Efficiency (fps/W)	3.8	12	-	6.4

[(1)] Normalized performance to 200 MHz frequency

Caffe framework at a frequency of 933MHz [45]. All inference experiments are done with batch size 1. The table shows that the proposed hardware accelerator overcomes Jetson Nano and Intel Movidius while running with Caffe framework, but Jetson has a better performance using TensorRT framework. The power consumption is 5W for Jetson and 3.92W for the proposed design. Unfortunately, Intel Movidius power consumption for GoogLeNet is not mentioned in the experiment. The power efficiency is the best for Jetson Nano using TensorRT framework with 12 frames/Watt,

TABLE 9. Comparison with other googLeNet hardware implementations.

	Zhao [19]	Gokhale [18]	Lu [46]	CoNNA_C3 [47]	This work
Platform	ASIC - TSMC	Zynq XC7Z045	Zynq ZCU102	Zynq ZCU102	Virtex-7 VC709
Max Frequency (MHz)	650	250	200	100	200
Precision	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed	12-bit fixed
Process Technology	65nm	28nm	16nm	16nm	28nm
Power (W)	0.859	9.48	23.6	-	3.92
Peak Performance (GOP/s)	242.4	116.5	257.4	17.325	129.2
Power Efficiency (GOP/W)	282	12.3	10.9	-	32.7
Power Efficiency (fps/W)	-	2.87	-	-	6.4
Performance (fps)[(1)]	23.6	27.2	-	4.95	25.1
Utilization efficiency	83%	91%	-	-	89%

[(1)] Normalized performance to 200 MHz frequency

but the proposed implementation is better while using Caffe framework with 6.4 frames/Watt.

Another comparison is made between the proposed accelerator and GoogLeNet hardware accelerators in the literature, as shown in Table 9. The first implementation is Zhao's hardware accelerator, which is an ASIC chip built with 65nm technology. The second implementation is Gokhale's FPGA implementation on Zynq XC7Z045. The last two implementations are Lu's implementation and CoNNA_C3 on Zynq ZCU102. The comparison is made between the implementations in terms of the operating frequency, fixed-point precision, process technology, power consumption, performance, and power efficiency, as shown in Table 9. The results show that the proposed accelerator provides the best performance in terms of the number of frames per Watt. In addition, it overcomes Gokhale's implementation in terms of peak performance and power consumption. Gokhale's implementation computes the number of frames per second for convolution layers only, so it processes 27.2fps compared to 25.1fps for the proposed implementation. In addition, Zhao's implementation overcomes the proposed accelerator in terms of GOP/s as it works on 650MHz. Also, it is an ASIC implementation, so the power consumption is expected to be lower than the FPGA implementations. While the other implementations use a plain GoogLeNet CNN model, the proposed implementation uses a compressed CNN model. This is one of the design advantages which improves the power consumption as discussed earlier.

The data access patterns variations in CNNs make it difficult for custom architectures to get higher utilization efficiency while processing all CNN layers. Consequently, utilization efficiency is stated as the ratio of the actual number of operations processed to the theoretical maximum number of the processed operations. This is translated to the ratio of actual fps to the utilization efficiency is 83% for Zhao's accelerator, 91% for Gokhale's accelerator, and 89% for this proposed work. Also, Zhao's and Gokhale's implementations compute the number of frames per second for convolution layer acceleration only, which is degraded when FC and average pooling layers are added.

VII. CONCLUSION

In this paper, a power-efficient convolutional neural networks accelerator based on GoogLeNet CNN was proposed. Weights pruning and quantization were applied, which reduced the memory size by 57.6× with a top-5 error rate of 2.6%. As a result, only FPGA BRAMs were used for weights and activations storage without using offline DRAMs. The compression model was explained in detail, and the reduction for every GoogLeNet layer was presented. In addition, this accelerator didn't use any DSP units as it replaced all multiplications by shifting operations. The accelerator was built based on a time-sharing/pipelined architecture that could process the CNN model layer by layer using 224 PEs. The architecture proposed a new data fetching mechanism that increased data reuse. All accelerator units were implemented in native RTL (Verilog). Moreover, several approximations and improvements were adopted to improve the design with a little accuracy loss. A word length of 12-bit was used after performing several experiments to select a suitable word length. The processor classified 25.1fps for GoogLeNet inference using 3.92W which was more power-efficient than the previous FPGA implementations for GoogLeNet CNN. It provided 49.5× power consumption improvement over Intel Core-i7 and 7.8× over NVidia GTX 1080Ti. Furthermore, the proposed hardware accelerator was compared with Jetson Nano and Intel Movidius. On the other hand, the processor achieved a top-5 classification accuracy of 91%, which was significantly higher than comparable architectures. Regarding future work, the control units in the proposed hardware accelerator can be reconfigured to process other CNN models such as ResNet or SqueezeNet. Also, the memory compression framework can be applied on the CNN software model to reduce the memory size and power consumption. In addition, the ASIC implementation can be made to get better performance in terms of power consumption, processing speed, and utilized area.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, no. 2, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[3] J. Mutch and D. G. Lowe, "Multiclass object recognition with sparse, localized features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2006, pp. 11–18.
[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[6] Y. Lee, H. Kim, E. Park, B. Yim, and H. Kim, "Optimization for object detector using deep residual network on embedded board," in *Proc. IEEE Int. Conf. Consum. Electron.–Asia (ICCE–Asia)*, Oct. 2016, pp. 1–3.
[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[8] S. Tamura, H. Ninomiya, N. Kitaoka, S. Osuga, Y. Iribe, K. Takeda, and S. Hayamizu, "Audio-visual speech recognition using deep bottleneck features and high-performance lipreading," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA)*, Dec. 2015, pp. 575–582.
[9] S. Ramos, S. Gehrig, P. Pinggera, U. Franke, and C. Rother, "Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1025–1032.
[10] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," 2015, *arXiv:1504.01716*.
[11] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," 2019, *arXiv:1901.06032*.
[12] J. Hoffmann, O. Navarro, K. Florian, B. Janßen, and H. Michael, "A survey on CNN and RNN implementations," in *Proc. IARIA*, 2017, pp. 33–39.
[13] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 5–14.
[14] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference on FPGAs: A survey," 2018, *arXiv:1806.01683*.
[15] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
[16] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, 2019.
[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[18] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[19] B. Zhao, J. Li, H. Pan, and M. Wang, "A high-performance reconfigurable accelerator for convolutional neural networks," in *Proc. 3rd Int. Conf. Multimedia Syst. Signal Process. (ICMSSP)*, 2018, pp. 150–155.
[20] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014, pp. 269–284.
[21] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.
[22] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL deep learning accelerator on Arria 10," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 55–64.
[23] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 25–34.
[24] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-S. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2016, pp. 16–25.
[25] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.
[26] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 40–47.
[27] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.
[28] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2010, pp. 247–257.
[29] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2009, pp. 53–60.
[30] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[31] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*.
[32] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. Inst. Radio Eng.*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[33] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C.-Z. Xu, "Dynamic channel pruning?: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–14.
[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[35] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNS," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.
[36] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*.
[37] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, Feb. 2020.
[38] K. Lee, S. H. Sung, D.-H. Kim, and S.-H. Park, "Verification of normalization effects through comparison of CNN models," in *Proc. Int. Conf. Multimedia Anal. Pattern Recognit. (MAPR)*, May 2019, pp. 1–5.
[39] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*.
[40] J. L. Holi and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Trans. Comput.*, vol. 42, no. 3, pp. 281–290, Mar. 1993.
[41] D. Larkin, A. Kinane, and N. O'Connor, "Towards hardware acceleration of neuroevolution for multimedia processing applications on mobile devices," in *Neural Information Processing* (Lecture Notes in Computer Science), vol. 4234, I. King, J. Wang, L. Chan, D. L. Wang, Eds. Cornell Univ., 2006, pp. 1178–1188. [Online]. Available: https://arxiv.org/abs/1806.01683
[42] (2019). *NVIDIA Jetson Nano*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano
[43] (2017). *Intel Movidius VPU*. [Online]. Available: https://www.movidius.com/
[44] (May 2019). *Running TensorRT Optimized GoogLeNet on Jetson Nano*. [Online]. Available: https://jkjung-avt.github.io/tensorrt-googlenet/
[45] (Jan. 2018). *Deploying Customized Caffe Models on Intel Movidius*. [Online]. Available: https://movidius.github.io/blog/deploying-custom-caffe-models/
[46] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 17–25.
[47] R. Struharik, B. Vukobratovic, A. Erdeljan, and D. Rakanovic, "CoNNA—Compressed CNN hardware accelerator," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 365–372.

AHMED J. ABD EL-MAKSOUD received the B.Eng. degree in electronics and electrical communications from Cairo University, Cairo, Egypt, in 2018, where he is currently pursuing the M.Sc. degree in electronic and communication engineering with the Faculty of Engineering. He was an ASIC Physical Design Engineer with Si-Vision, Egypt. His research interests include digital electronics, FPGAs and ASICs, parallel computing, and computer architectures.

MOHAMED EBBED received the B.Eng. degree in computer engineering degree from Cairo University, in 2021. He is currently pursuing the M.Sc. degree in informatics with the Technical University of Munich, Munich, Germany. His research interests include computer vision, models optimization, and reinforcement learning.

AHMED H. KHALIL received the B.Eng. degree in electronics and electrical communications engineering (EECE) from Cairo University, Cairo, Egypt, in 1983, and the M.Sc. and Ph.D. degrees from the EECE Department, Faculty of Engineering, Cairo University, in 1987 and 1992, respectively. He has been in the electronics and communications industry for over 32 years of hands-on practical knowledge. He is currently a Professor with the EECE Department, where he has been a Faculty Member, since 1983, and the Vice Director of the Design Laboratory for Electronics and Communication Systems (DLECS), since 2006. He also teaches several courses on analog and digital electronics and signal processing at Cairo University and other universities and institutes in Egypt. His research interests include signal processing, embedded systems, and FPGA-based systems using broad range of tools and technologies on diverse types of platforms with solid track record of supervising many mega projects and experience in electronic design and technology integration.

HASSAN MOSTAFA (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in electronics engineering from Cairo University, Cairo, Egypt, in 2001 and 2005, respectively, and the Ph.D. degree in electrical and computer engineering from the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2011. He was an NSERC Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. He is currently an Associate Professor with the Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, Giza, Egypt, and he was on leave from the Department of Electronics and Electrical Communications, Cairo University. His postdoctoral work includes the design of the next generation FPGA in collaboration with the Fujitsu Research Labs in Japan/USA. He has authored/coauthored more than 250 papers in international journals and conferences and has authored/coauthored five published books. His research interests include neuromorphic computing, the IoT hardware security, software-defined radio, reconfigurable low power systems, analog-to-digital converters, low-power circuits, subthreshold logic, variation-tolerant design, soft error tolerant design, statistical design methodologies, next generation FPGA, spintronics, memristors, energy harvesting, MEMS/NEMS, power management, and optoelectronics. He has been a member of the IEEE Technical Committee of VLSI Systems and Applications, since 2017.