Towards Full Stack Acceleration of Deep Convolutional Neural Networks on FPGAs

Shuanglong Liu, Hongxiang Fan, Martin Ferianc, Xinyu Niu, Huifeng Shi, and Wayne Luk, *Fellow, IEEE*

***Abstract*—Due to the huge success and rapid development of convolution neural networks (CNNs), there is a growing demand of hardware accelerators that accommodate a variety of CNNs to improve the inference latency and energy efficiency, in order to enable their deployment in real-time applications. Among popular platforms, Field-Programmable Gate Arrays (FPGAs) have been widely adopted for CNN acceleration because of their capability to provide superior energy efficiency and low-latency processing, while supporting high reconfigurability, making them favourable for accelerating rapidly evolving CNN algorithms. This paper introduces a highly customized and streaming hardware architecture which focuses on improving the compute efficiency for streaming applications by providing full stack acceleration of CNNs on FPGAs. The proposed accelerator maps the most computational functions, i.e. convolutional and deconvolutional layers in one unified module, and implements the residual and concatenative connections with high efficiency, in support to the inference of mainstream CNNs with different topologies. This architecture is further optimized through exploiting different level of parallelism, layer fusion and fully leveraging DSPs. The proposed accelerator has been implemented on Intel's Arria 10 GX1150 hardware and evaluated with a wide range of benchmark models. The results demonstrate a high performance of over 1.3 TOP/s of throughput, up to 97% of compute (MAC) efficiency, which outperforms the state-of-the-art FPGA accelerators.**

***Index Terms*—Convolutional Neural Networks (CNNs), Field Programmable Gate Arrays (FPGAs), Hardware Accelerator, Unified Architecture, Layer Fusion, Deep Learning.**

I. Introduction

RECENTLY, large and deep convolutional neural networks (CNNs) have become widely adopted in many tasks such as image classification [1], [2], object detection [3] and semantic segmentation [4]. Particularly, they have been deployed in a variety of real-life and real-time applications such as smart cities, cameras and remote sensing [5]. In these applications, CNNs have shown great accuracy improvement in comparison to traditional machine learning (ML) algorithms due to their high learning ability and their structural similarity to the visual cortex of human brains. However, most successful CNN models exhibit very high computational complexity, and require vast memory and processing power.

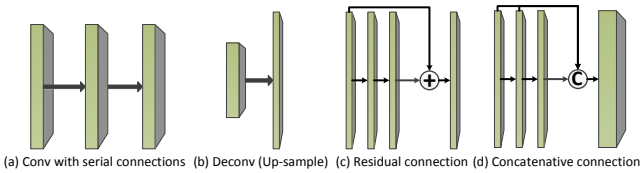

Fig. 1. The main functions and connectivity in CNNs: (a) standard convolutions with serial connection; (b) deconvolution, also known as *up-sampling*, which extrapolates new information from the input feature map and is widely used in segmentation models; (c) residual connection, also called as *shortcut* connection, where the results of one layer skip one or more layers and then are added to subsequent layers at different depth levels; (d) concatenative connection, which is a utility layer that concatenates its multiple input blobs to one single output blob onto which no mathematical functions are performed. Unlike previous FPGA-based accelerators targeting CNN types shown in (a), our work aims to improve the efficiency of CNNs of all these types.

The operations that compose a CNN are not well-suited to the Von Neumann computer architecture at the heart of CPUs. They are better suited to hardware architectures with distributed, massively-parallel computation and local memory such as graphics processing units (GPUs) or Field-programmable Gate Arrays (FPGAs). In particular, GPUs with highly parallel architectures can achieve high throughput on CNNs by processing parallel samples in batches. The efficiency of GPUs relies largely on the regularity of data and batch size, which works well for off-line training but not in practice while targeting real-time inference [6]. For example, images in streaming applications arrive one by one and using batch processing can greatly increase latency, which is critical to the system's performance.

Designing a dedicated hardware for accelerating CNNs requires significant investment and time to develop. However, the ML community keeps to rapidly evolve CNNs. For example, VGG16 [2] was first introduced in 2014 for object detection, one of the most popular tasks in computer vision, which employed a uniform convolutional kernel size with serial layer connectivity. A year later, CNNs have been in the trend of employing residual (ResNets [7]) and concatenative connections (GoogLeNet [8]), which introduce irregular connectivity across layers. Moreover, networks such as YOLOv3 [9] employ both types of irregular connections, making potential accelerators e.g. for VGG16 already obsolete. These irregular connections are shown and explained in Figure 1.

Apart from object detection, semantic segmentation has been widely studied across a variety of application domains, in order to provide pixel-wise information of the image.

This work was supported in part by the United Kingdom EPSRC under Grant EP/L016796/1, Grant EP/N031768/1, Grant EP/P010040/1, Grant EP/L00058X/1 and Grant EP/S030069/1, and in part by the funds from Corerain, Maxeler, Intel, Xilinx and SGIIT. *(Corresponding author: Shuanglong Liu.)*

S. Liu, H. Fan and W. Luk are with the Department of Computing, Imperial College London, London, SW7 2AZ, UK.

M. Ferianc is with the Department of Electronic and Electrical Engineering, University College London, London, UK. This work was done when he pursued his master degree at Imperial College London.

X. Niu is with Corerain Technologies Ltd., Shenzhen, China.

H. Shi is with the State Key Laboratory of Space-Ground Integrated Information Technology (SGIIT), Beijing, China.

E-mail: s.liu13@imperial.ac.uk

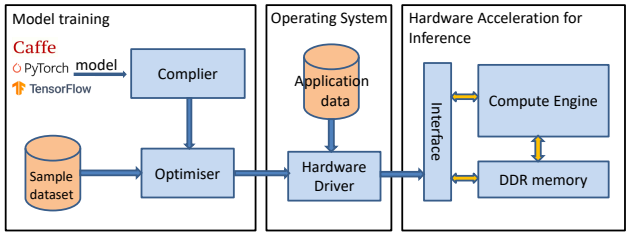

Fig. 2. An overview of the proposed optimization framework to accelerate the mainstream CNN models for low-latency inference.

Deconvolution layer (Deconv), also named as up-sample in literatures, is hence introduced in models such as SegNet [4] or U-Net [10], in addition to classic 2-D convolution (Conv), which also employ concatenative connections. In these models, Deconv together with Conv layers make up the majority of computation [11]. As a result, they are far more computationally intensive than models designed for image classification or object detection. Therefore, a general customisable hardware architecture, without the need to develop dedicated circuits, with the capability to support all kinds of models mentioned above is crucial for rapid system development. This requires special focus and efforts on the compute efficiency of both Conv and Deconv layers, as well as the irregular connections.

To address the challenges of CNNs with irregular shapes and/or Deconv layers, and adapt to varying and evolving CNNs, we propose a full stack optimization framework and develop a hardware accelerator based around FPGAs. Figure 2 presents the workflow of the proposed method in three parts. In the training stage, the compiler tool accepts a newly designed and trained CNN model from ML frameworks such as Tensorflow [12], PyTorch [13] or Caffe [14]. Then the compiler generates an optimized representation, i.e. a streaming graph, which can then be run on hardware. In application level, the execution instructions are generated for the already trained CNN model and then the system makes calls to the computational engine on the FPGA for inference. The whole network inference is executed on the hardware engine with the CNN's weights and input image stored in DDR memory.

In the heart of our approach is a compute engine, i.e. the hardware accelerator which aims to improve the compute efficiency and reduce the inference latency for the mainstream CNNs with different topologies. It builds on a unified hardware architecture which maps both Conv and Deconv layers into a single hardware module, in order to improve the resource efficiency. Besides, the streaming accelerator maps the irregular connections (residual and concatenative connections) with high efficiency by organizing the hardware blocks in a way where all blocks are kept busy at all times, also by using a custom-tailored design of smart cache system. Additionally, the accelerator is further optimized through exploiting different level of parallelism and fully leveraging the digital signal processing blocks (DSPs) on an FPGA. Finally, the CNN is quantized through 8-bit fixed-point quantization scheme [15] to achieve higher performance without loss of accuracy.

The novel contributions of this work are as follows:

- Automated acceleration framework, which enables users to employ the trained network models on FPGAs with optimal hardware configurations;
- Unified hardware architecture designed on the matrix multiplication module to implement the main computation layers with high compute efficiency, specifically Conv and Deconv with arbitrary kernel size;
- Streaming accelerator with efficient mapping of residual and concatnentative connections and optimized with techniques such as input reshaping and layer fusion; Therefore, it can support the mainstream CNNs with regular or irregular structure more efficiently;
- Latency estimation method using Gaussian process regression to predict the latency in real FPGA implementation more accurately, which in turn reduces the design time for trade-off between accuracy and performance, and improves the hardware design productivity;

Leveraging all these advances into a single system, we have built an efficient CNN inference engine on an FPGA (Intel's Arria 10) with high compute efficiency. The compute efficiency is measured using the number of useful multiply-accumulate (MAC) cycles taken by the DSP blocks compared with the maximum peak performance. Achieving a high compute efficiency across a wide range of CNN models is a challenge for many hardware accelerators. The high compute and energy efficiency of the proposed design comes from the high resource utilization (DSPs and local memories) and efficiency due to several factors:

- Ability to exploit over 97% of the DSPs in the FPGA device with flexible DSP configurations (Section VI-D);
- Ability to occupy the DSPs during most of the time ($> 90\%$) by: 1) implementing the main computation operations in one unified architecture (Section III-A); and 2) reducing communication time with efficient execution of irregular connections (Section III-D) and layer fusion method (Section IV);

II. Background and Related Work

In this Section, we first review recent advances for efficient CNNs in both algorithm and hardware implementations. Then, we summarize the limitations of previous FPGA-based accelerators for CNNs in comparison to our design. Quantitative evaluation and comparison will be presented in Section VI-F.

A. CNN Layer Overview

CNNs are built of several computational operations stacked on top of each other, commonly known as layers, and most modern networks have residual or concatenative connections between them. Frequently used layers are 2-D convolutional (Conv), up-sampling (Deconv) or fully-connected (FC) layers. These three layer types take up over 90% of computation in a CNN model. Besides, there are pooling and batch normalization [16] layers or activations such as rectified linear unit (ReLU).

As illustrated in Code 1, the Conv or Deconv receives $C \times H_i \times W_i$ sized input feature maps, and then these inputs are convolved or deconvolved with a kernel with the shape of $F \times C \times K \times K$. Each kernel window with the size of $K \times K$

Code 1 Convolution and Deconvolution Algorithms

Input: Input feature map $\mathbf{I}$ of shape $C \times H_i \times W_i$;
Weight matrix $\mathbf{W}$ of shape $F \times C \times K \times K$;
Output: Output feature map $\mathbf{O}$ of shape $F \times H \times W$;

1: for $(f = 0; f < F; f++)$ **// filter loop**
2: for $(c = 0; c < C; c++)$ **// channel loop**
3: for $(h = 0; h < H; h++)$ **// row loop**
4: for $(w = 0; w < W; w++)$ **// column loop**
5: // Conv:

$$\mathbf{O}[f][h][w] += \sum_{i=1}^{K-1}\sum_{j=1}^{K-1} \mathbf{W}[f][c][i][j] * \mathbf{I}[c][h*S+i][w*S+j]$$

6: // Deconv:
$\mathbf{O}[f] += deconv(\mathbf{I}[c], \mathbf{W}[f][c])$ // as shown in Figure 3.

is performed with one channel of the input ($H_i \times W_i$) by sliding the kernel with a stride of S to produce one output feature map ($H \times W$); then the results of C channels are accumulated to produce one channel of output (channel loop in line 2). All filters of the output feature maps ($F \times H \times W$) are generated by repeating this process F times (filter loop in line 1). Line 5 of Code 1 describes the 2-D convolution. Deconv layers are implemented as transposed convolutions in CPUs or GPUs [17]. Before performing the transposed convolution, zeros need to be inserted into the original input feature maps. FC layers can be converted to a Conv layer by considering the kernel size K. For example, an FC layer with the input size of $C \times H \times W$ and the output size of $F \times 1$ can be implemented as a Conv layer with the kernel size of $F \times C \times H \times W$.

B. Efficient CNNs

It has been a general trend of increasing the depth of CNNs using residual and concatenative connections between their layers, to improve their classification accuracy as well as the speed of training [18]. As a result, the networks are gradually becoming structurally denser and thus more complex, which largely limits their application in resource constrained settings, such as in edge devices for Internet-of-Things (IoT) applications. Therefore, many research teams have proposed methods to reduce the computation complexity of CNNs both at algorithm and hardware implementation levels.

Novel algorithms including Winograd convolution [19] or fast Fourier transform (FFT) [20] focus on compute reduction techniques. FFT performs the convolution operation in frequency domain, thus it turns the originally space domain operation into a Hadamard product between the input and the convolution kernel. Winograd convolution computes minimal complexity convolution over small tiles, which reduces the number of multiplications by a factor of approximately 2.25× using the filter $F(3\times3, 2\times2)$ [21]. Other algorithmic advances cover model compression which shrinks model representation by channel pruning or resolution multipliers, used for example in MobileNet [22]. Another technique to compress models is to replace standard convolutions by depth-wise separable Conv [23], which reduces the computation by a factor of K^2.

From the perspective of hardware level, researchers [24]–[26] have proposed: (1) a quantization method, which captures the speciality of FPGAs with capability of custom precision support to save computation resources; (2) loop unrolling strategies for multiple parallel processing. A further step into improving quantization involves binarization for both weights and data, while executing the CNN on FPGA, since in this case the multiplication can be simply implemented as a XNOR gate [27]–[29], which largely relieves the use of limited DSP resources in current FPGAs. Others have proposed residual binary inputs and weights to improve binarized networks, which can improve the accuracy while still maintaining almost the same computing resources in hardware [30].

C. Related Work

Recently, various FPGA-based accelerators for CNN inference have been proposed with the key objectives of designing a system with high energy efficiency and low latency. These accelerators, however, are generally targeting relatively structurally simple networks such as AlexNet [1] or VGG16 [2]. The common strategy used among these accelerators is to minimize the data and weight movement from the off-chip memories to the compute engine which is implemented with FPGA's fabric. The techniques include (1) double buffer, to overlap the computation time and the data/weight load time [24], [31]; and (2) layer fusion, to process multiple CNN layers in a pipelined manner, allowing for instant use of intermediate data without external memory access [18], [26]. However, a majority of prior accelerators only focus on Conv layers, thus provide high efficiency only for CNNs with regular shapes.

Few works have studied the acceleration of Deconv layers and generative networks (GANs) [32]–[34]. However, these works focus specifically on accelerating Deconv in GANs which consist solely of deconvolutional layers. Therefore, they did not attempt to accelerate other models such as those used in segmentation models which employ both Conv and Deconv layers. Our prior work [11] optimized the operations of both Deconv and Conv layers for semantic segmentation. An approach was proposed to address the compute inefficiency incurred by the sparsity of Deconv when implemented as transposed Conv. However, two different hardware modules are deployed for Deconv and Conv separately in this design and their DSPs for multipliers are not shared, which caused the inefficiency of resource utilization.

Moreover, previous FPGA-based accelerators did not efficiently support models with irregular connections. Venieris *et al.* [35] designed three hardware blocks for each irregular network connection for networks that they evaluated, i.e. GoogLeNet, ResNet-152 and DenseNet-161. This approach can be a solution for a reconfigurable FPGA design, but it still leads to low resource efficiency in execution and reduction in the design's productivity. McDanel *et al.* [18] introduced a network without any concatenative or residual connections with small accuracy loss. Although competitive performance on ImageNet [36] can be achieved in the mobile setting, it did

not solve the problem from the hardware perspective, and in other applications, users are gradually more inclined towards using residual or concatenative connections.

Compared to previous work, we first implement the main computation layers among CNN models, i.e. convolutional, deconvolutional and fully-connected layer and map them into a single unified module which improves the MAC efficiency during inference. Second, the proposed accelerator supports both residual and concatenative connections for a general set of networks with only one element-wise residual hardware block, and a high compute efficiency for these irregular structures is achieved through designing a smart memory system (Section III-D). As a result, our approach provides high compute efficiency for both regular and irregular network structures without the need to reconfigure the FPGA fabric.

III. Streaming Accelerator Architecture

This Section first proposes a unified architecture to support the main operations: convolution, deconvolution and fully connected layers in CNNs, which serves as the key hardware module in our compute engine. We explore different levels of parallelism for the unified architecture as well as the overall accelerator. Then, it presents the overall structure of the accelerator with a smart cache design that allows the implementation of residual and concatenative connections, while maintaining high efficiency for a general set of CNN models. Lastly, it employs the 8-bit fixed-point quantization scheme [15] in this work.

A. The Unified Architecture

The direct mapping of CPU- or GPU-based Deconv algorithm, i.e. transposed convolution on FPGA will incur the compute inefficiency due to the zero insertions leading to meaningless multiplications with zeros. In this work, an efficient 2-D Deconv approach proposed in [11] is used in our hardware implementation, as illustrated in Figure 3. This approach multiplies input pixels with the corresponding weight kernel and sums the overlapping area in output maps. It improves the compute efficiency by exploiting the sparseness of transposed convolutions. With this method, the total number of multiplications in Deconv is reduced from $K^2 \cdot F \cdot C \cdot H \cdot W$ to $r \cdot F \cdot C \cdot H \cdot W$ where $r \in [1, 2)$.

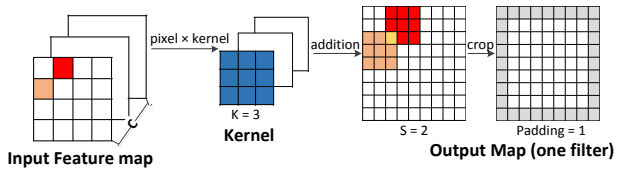

Fig. 3. Visualization of our approach to implement the Deconv layer with $K = 3$, $S = 3$, $Padding = 1$.

Existing FPGA-based accelerators such as [11], [37] implement 2-D Conv by unrolling the dot-product loop in line 5 of Code 1 using $K \times K$ multipliers. However, this type of architecture cannot be reused for the Deconv approach mentioned above because of the different computing pattern.

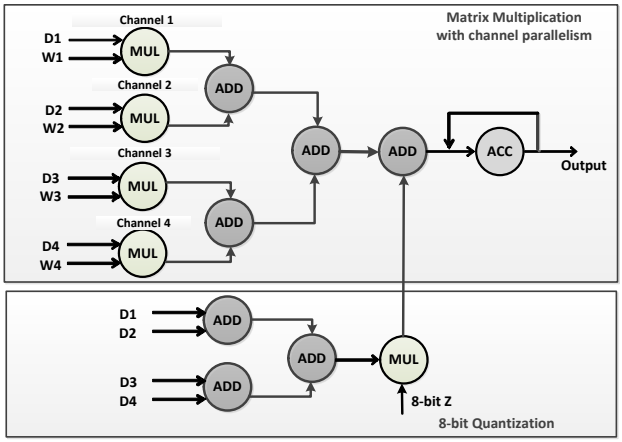

Fig. 4. The unified architecture proposed to map Conv, Deconv and FC on FPGA with parallel channel processing and the 8-bit quantization module to support integer-only arithmetic inference [15].

Besides, it is difficult to reconfigure the compute kernel back for Conv with multiple kernel sizes (such as 3×3, 5×5 or 7×7).

To improve the resource efficiency, we propose a unified accelerator architecture to implement both Conv and Deconv with an arbitrary kernel size. FC layers are always performed as a Conv layer without the need to introduce additional blocks, and thus achieve the highest resource occupancy during runtime. In this architecture, each multiplier is responsible for computing a single output pixel, such that the 2-D Conv or Deconv is performed in a single multiply and accumulate (MAC) unit for one output. As shown in Figure 4, it consists of a matrix multiplication (MM) module which computes multiple channels of input in parallel, and a quantization module which computes the sum of the input pixels, to support the 8-bit linear quantization scheme proposed in [15]. The quantization scheme for CNNs achieves a very high compute density without observing loss of accuracy, as we will show in Section VI-E. The details on the implementation of Conv and Deconv are explained as below.

Conv: To compute one output pixel, the corresponding $K \times K$ input pixels of the feature maps and $K \times K$ weights are sequentially multiplied in a single multiplier. Then, the multiplied results are flowed into an accumulator (ACC shown in Figure 4) for accumulation to compute one output pixel of the input maps of one channel. Therefore, one output of convolution requires $K \times K$ hardware cycles in total.

Deconv: The Deconv approach shown in Figure 3 is more complex than Conv in terms of hardware implementation. The number of MAC operations required depends on the position of the computed output pixel, since there are different overlapping rows and columns presented in the output map in Figure 3. In total, three cases are to be considered: (1) for the output with non-overlapping rows or columns, only one input pixel is multiplied by the weights; (2) for the output with only one overlapping row or column, two adjacent input pixels in row or column dimensions are sequentially multiplied by the corresponding weights; (3) for output with both overlapping row and column, four adjacent input pixels in row and column dimensions are sequentially multiplied by the weights. For

the last two cases, the multiplied results are flowed into the accumulator for accumulation. Hence, 1, 2 or 4 clock cycles are required respectively to compute one Deconv output of one channel input maps in the three cases.

Therefore, the architecture can implement convolutions with any kernel size and strides as well as deconvolutions. It is also capable of supporting other convolution-based operations in CNNs such as 1-D Conv or dilated Conv, by feeding the data and weights into the multipliers in the right sequence.

B. Parallelism Exploration

We explore different levels of parallelism in order to improve the resource utilization and compute efficiency of our accelerator. Three levels of parallelism can be utilized for parallel processing in convolution-based operations: filter parallelism, channel parallelism and data parallelism. They correspond to unrolling the loops in lines 1, 2 and 3 of Code 1 respectively. Data parallelism is utilized in previous designs such as [11]. However, the employment of data parallelism will result in computational inefficiency in practical hardware design due to the following factors:

1) **Workload imbalance** when performing Deconv. When employing data parallelism, it computes multiple output pixels in one row of the output feature maps in parallel. However, Deconv has 3 separate modes with respect to which the output in one row can be produced, just as we have mentioned above. As a result, the workload of the multipliers is imbalanced and some multipliers must be kept idle to wait for others to finish processing, resulting in low multiplier utilization.

2) **Inefficiency** when the input width of a layer cannot be divided by the degree of data parallelism. The degree of parallelism in hardware must be a fixed number, e.g. 32. However, the layers in CNNs often have the input maps with different heights and widths, and it is impossible to have a degree of parallelism in which all the widths of layers in the network are fully supported. For example, for $W = 36$, the compute efficiency is only $\frac{36/32}{\lceil 36/32 \rceil} = 56\%$. This inefficiency can be relieved by batch processing, but as previously mentioned, it increases the latency for streaming applications.

Therefore, instead of using data parallelism, we employ the channel parallelism in this work, as already shown in Figure 4. Multiple channels (PC) of inputs are multiplied with the weights in parallel and the results are then added together using an adder tree before the accumulation. The advantage of this design is that each multiplier's workload is balanced, since the output pixels of different channels have the identical position in the output maps. Additionally, the layers' input channel (except for the first layer) in CNNs are often a power of two, or can be tuned to a power of two, so that they can be divided by the degree of channel parallelism (PC) which is normally a power of two as discussed previously. Hence, the channel parallelism does not lead to any loss in the utilization of multipliers and it adapts to the algorithmic design from users. On the contrary, once the size of the first layer's input is determined, the size of all other layers are automatically decided, while the channel numbers are independent among layers in one CNN model.

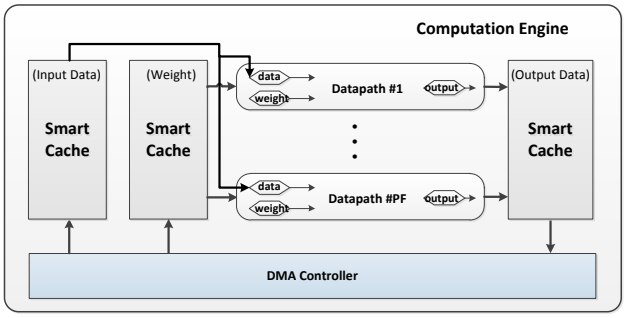

Fig. 5. The architecture of the overall accelerator which supports both channel parallelism and filter parallelism with double buffer technique employed.

Furthermore, the accelerator supports parallelism in the filter dimension (PF). The compute engine instantiates PF datapaths in total, where each datapath includes a MM module with channel parallelism (PC) and other hardware blocks, that map a CNN onto an FPGA with parallel processing power. The overall design is shown in Figure 5. Weights of the CNN are read from the DDR memory and cached in weight buffers using double buffer technique to overlap the load time. Intermediate results are read from and written back to the local caches directly on the FPGA, with smart read/write controls. Note that the PF datapaths share the identical quantization module in Figure 4 as they use the identical pixels as input data, which largely saves the hardware resources. The architecture design of the complete datapath is presented in Section III-C while the smart cache system is described in Section III-D.

C. Hardware Building Blocks

Figure 6 shows the hardware blocks of one datapath that map a CNN model onto the FPGA. Each datapath consists of a MM module, i.e. the unified compute cell to perform the convolution, deconvolution or fully-connected operation. It is then followed by a ReLU module which performs the non-linear activation such as ReLU or leaky ReLU. The following pooling module is added to run average or maximum pooling on the input data. The residual block accepts one input from one of the preceding hardware blocks which computes the results of the current layer, and another input from the local cache which stored the result from the previous operation. It can perform element-wise operations such as *add*, *subtract* or *multiply*. The operations are implemented using the available DSPs and one DSP can be configured to run any of the three kinds of operations above during runtime simply by using control signals. Therefore, our design does not introduce any additional resource overhead by supporting three types of element-wise operations instead of potentially only supporting *multiply* in the residual block. The final connected block is a global average pooling (GAP) module, which is usually employed before a final fully-connected layer in a CNN [38]. The concatenative layers do not perform any operations, thus no hardware block needs to be instantiated and they are actually implemented through smart cache design, as their operation is mainly dependant on routing of the incoming data.

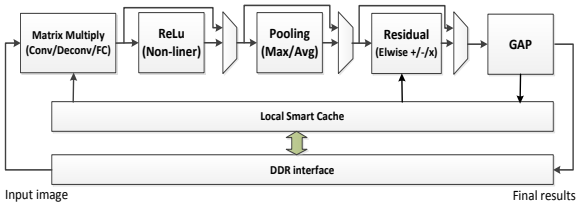

Fig. 6. The hardware blocks instantiated in our accelerator to map the mainstream CNNs on FPGA.

The trick in the datapath design is that each of the hardware blocks can be bypassed through multiplexers which enable flexible layer configurations. Thus, our design is capable of implementing a wide range of CNN topologies such as GoogLeNet [8], ResNet [7], VGG16 [2] and YOLOv3 [9] by simply correctly configuring the datapath through control signals that influence the information flow. Besides, the whole datapath is run in a pipelined manner and all other functionalities can be run in parallel in the main module, thus keeping these blocks busy during most of the execution time and achieving high resource efficiency across the CNN models.

The input image and weights are stored in a DDR memory. While processing, they are first cached in the on-chip Block RAMs (BRAMs), i.e. local cache on the FPGA, then all the intermediate results are stored in the local cache without accessing the DDR memory, to avoid additional communication cost. The final result after execution is stored back to the DDR memory for further evaluation in the CPU. Therefore, the performance of our system is not limited by the bandwidth of the DDR interfaces.

D. Smart Cache Design

One of the advantages of FPGAs in comparison to GPUs and CPUs is their large on-chip bandwidth, since the local BRAMs can be customized with large data width to decrease the access latency for the frequently used data. For example, in convolution, each input pixel is reused $K \times K \times F$ times, and weights are only used once. Data buffers are also needed to cache the input and output of standard convolutions, inputs of residual block, and multiple input blobs of concatenative connections. Therefore, efficient utilization and management of the local caches on the FPGA is crucial to the performance of the overall system. Here, we introduce our smart cache design, in order to achieve the maximum memory utilization while maintaining parallel processing capabilities in channel and filter dimensions. Besides, we show how the local cache is divided and balanced into different parts, in order to improve the efficiency of concatenative and residual connections.

Data Storing Pattern in Cache. The storage data pattern in caches should mainly consider the support of parallel processing. Weights are simple and straightforward to be cached. Weight buffers are divided into PF memory banks and each bank stores one set of filter weights, i.e. $C \times K \times K$. Each bank has a memory width of PC weights with the depth of $K \times K \times TC$, where $TC = C/PC$. As shown in Figure 7, the weights are stored in channel dimension first, followed by width and height dimensions. Weights of multiple filters are fed into different rows of the datapaths in parallel.

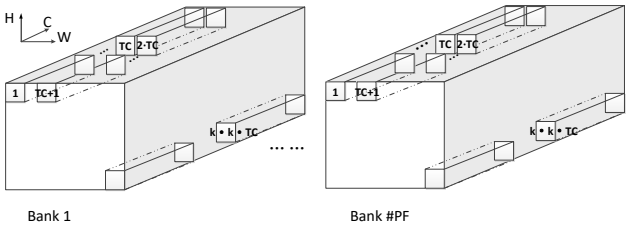

Fig. 7. Weight storing pattern in the weight buffer which consists of PF banks and each bank stores one set of filter weights with the width of PC weights (8×PC bits) and the depth of $K \times K \times TC$. Numbers in the Figure represent the memory address of the attached data group.

Data buffer design is much more challenging. The feature maps can be stored in the data buffer in two orders: channel-major and block-major. Both methods store the input feature maps in one single memory with the width of PC data pixels in channel dimension for implementation of channel parallelism. The illustration of both methods is shown in Figure 8. **Channel-major:** it stores the data pixels in the order of $H \times W \times C$. The data in channel dimension are stored first, then followed by width and height. **Block-major:** it stores the feature maps block by block, as the total volume can be regarded as TC blocks in the channel dimension. Each data block is stored in the order of $H \times W \times TC$. In this work, we use the block-major storing method for more efficient implementation of concatenative connections, as we will discuss later.

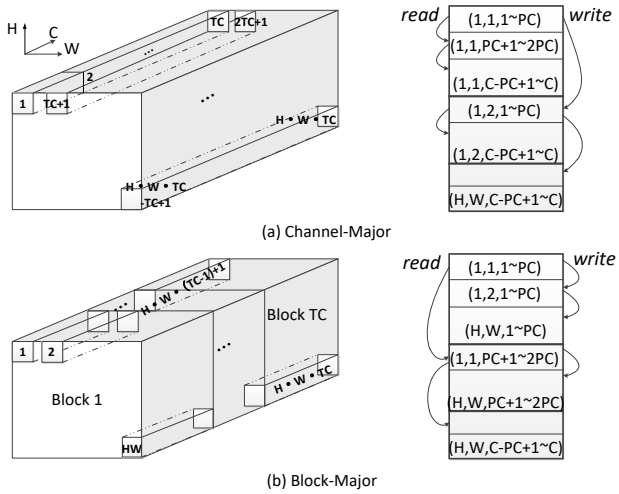

Fig. 8. Two alternatives of the data storing pattern in the data buffers, and their corresponding reading and writing patterns with channel-major computing.

Computing Pattern. Corresponding to the data storage pattern, there are two alternative ways for computing the standard Conv. For convenience, we still name them as channel-major and block-major computing patterns. **Channel-major:** it computes the final result of one data point first, thus it needs to access the data in one $K \times K$ window along the channel dimension until the end. This results in the datapaths reading $K \times K \times C$ input pixels in $TC \cdot K \cdot K$ cycles and then generating PF output pixels. The datapaths share the

same input pixels. Every $TC \cdot K \cdot K$ cycles, the accelerator generates PF results and in total $TC \cdot K \cdot K \cdot H \cdot W \cdot F/PF$ cycles are needed. When block-major storage is used, the data are read discontinuously in the input buffer. **Block-major:** it computes the results in width and height dimensions first instead of channel dimension, and the intermediate results of one block size need to be cached during the process. Every $K \cdot K \cdot H \cdot W \cdot TC$ cycles, it generates the results of the whole maps of PF filters, i.e. $H \times W \times PF$ output pixels. Compared to the channel-major method, it is more efficient for the DDR memory access since it generates a large volume of data consecutively and thus enables burst transfers of results to the DDR memory. However, it needs large buffers to cache the intermediate results with the size of $H \times W \times PC \times PF$, which increases the overhead of local caches.

Nevertheless, the channel-major computing does not need any cache for intermediate results and provides us with more efficient utilization of local memories. Therefore, it better suits our architecture in which all the layers are processed by using on-chip memories. The behaviours of different combinations of storing and computing patterns in cache design are summarized in Table I. In this work, the **block-major storing** and **channel-major computing** are utilized with comprehensive consideration of design complexity and compute efficiency. Note that the output results are always produced in the block-major pattern, because the results are generated in the filter dimension, so the block-major storing will lead to continuous writing behaviour, as shown in Figure 8.

TABLE I
A SUMMARY OF THE BEHAVIOURS OF STORING AND COMPUTING PATTERNS IN CACHE DESIGN

Behaviour: Storing / Computing	**Channel-major**	**Block-major**
Channel-major: × DDR burst transfer √ no cache overhead	√ continuous read × continuous write × concat. efficient	**This Work:** × continuous read √ continuous write √ concat. efficient
Block-major: √ DDR burst transfer × no cache overhead	× continuous read × continuous write × concat. efficient	√ continuous read √ continuous write √ concat. efficient

Data Buffer Organization. This part mainly considers how to manage and balance the data buffers for standard convolution, residual and concatenative connections. When performing standard Conv, two data buffers are needed which are input and output buffers respectively. Residual connection has two data inputs and a single output, while concatenative connection can have more than two inputs and again a single output. As shown in Figure 9, when connecting inputs in the residual layer, the input maps are stored in the input buffer and the accelerator can run standard Conv first, then its output is connected to the residual block with the other input coming from a second data buffer - *Residual Buffer* and finally, the residual output is stored in the output buffer. With this design, we can keep both MM module and residual block busy at the same time which guarantees high resource occupancy.

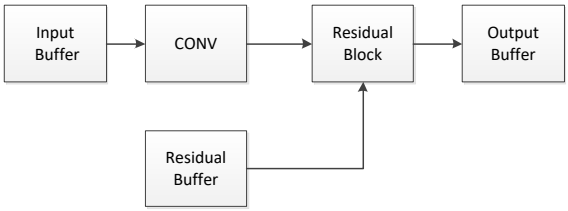

Fig. 9. A simplified illustration of the implementation of residual connections.

Since there are usually more than one residual or concatenative connections in a single network, either memory buffer can be used as an input or output or residual buffer. Therefore, in this work, we customize three memory buffers to cache the data, that all have the identical size and structure. When performing the concatenative connections, because the data are stored in buffers per block, they are concatenated together just by jumping to the other memory location which stores the other input blob. This also works for even more intricate input patters, such as three input blobs concatenated, by storing the multiple data blobs in one memory buffer. The simplicity and efficiency of implementation of the concatenative connection is owed to the chosen block-major storing pattern in data buffers.

E. Overall Accelerator

The overall system is shown in Figure 10. It consists of the host processor, computation engine, on-/off-chip interconnect (DMA) and off-chip DDR memory. The host processor is used to configure the parameters of layers when running the CNN model in the computation engine. All the weights of the model, the input image and the final results of classification/detection/segmentation are stored in the DDR memory.

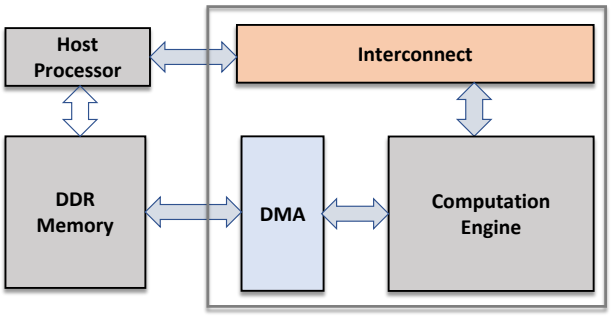

Fig. 10. An overview of the whole hardware system for FPGA-based acceleration for CNN inference.

IV. Design Optimizations

This Section presents the optimization techniques used in our approach to improve resource efficiency and compute efficiency of the proposed accelerator.

A. Layer Fusion

Layer fusion is a common optimization technique to minimize data movement which is considered in practical designs. By storing all intermediate data in on-chip memory, layer fusion processes multiple CNN layers at the same time in

a pipelined manner without the need for external memory access. As illustrated in Figure 5, input and output feature maps are cached in the data buffers using BRAMs during the execution. The memory size and data structure of these data buffers are the same, as described above. Before the processing of the first layer, the input data are transferred from the DDR and cached in one buffer. Then, the inputs are streamed into the datapaths, while the outputs are simultaneously flowing into the second data buffer. When the computation of the first convolution finishes, the second data buffer acts as the input buffer for the second convolution and the outputs will be cached in the other buffers. At the end, the final outputs are transferred back to the DDR from the data buffers. Double buffer technique is also used to cache weights, in order to overlap the weight load time with the computation time. For simplicity, it is not shown in Figure 5.

B. Input Reshaping to Improve Utilization

For CNNs trained on ImageNet [36], nearly 10-15% of the total computation is associated with the first convolution layer because of the large spatial size of the input image [18]. However, the computation of the first convolution layer has not been mapped well onto the previous hardware accelerators such as those based on the systolic architectures [39], because the input image only offers a small number of channels, which cannot fully utilize the input bandwidth and leads to the under-utilization of the computing resources.

To solve this imbalance, we propose to reshape the first layer to improve the resource utilization of our design. The input maps are divided into multiple small blocks. Then, we concatenate these blocks together along the channel dimension. Correspondingly, in order to fit them into our computation engine, the multipliers of the MM module shown in Figure 4 are grouped by 4, and each group takes three channels of data as input and generates one output. Hence, each datapath generates PC/4 output pixels in total at a time. With the help of the proposed input reshaping, the compute efficiency of the first layer is increased from 3/PC to 3/4=75%.

C. DSP Configuration

The theoretical performance of the system depends on the number of multipliers used in our design. In FPGAs, DSPs are often used to implement multipliers, which makes them the most limiting resource for CNN acceleration. Owing to the 8-bit quantization scheme, our accelerator has the potential of a very high compute density.

In Intel Arria 10 FPGAs, the DSP blocks can be fractured into two 18 $\times$ 19 integer multipliers. However, even using one DSP to implement two fixed-point multipliers is still a waste of resources because the 18 $\times$ 19 multiplier is implemented only for one 8 $\times$ 8 multiplication. To fully leverage the DSPs and logic elements (ALMs), we propose to use one 18 $\times$ 19 multiplier together with some ALMs to implement two 8 $\times$ 8 multiplications and one 16-bit addition. Also, to prevent the ALMs excessing the constraints of the device, we use DSPs in two configurations: 1) one DSP plus ALMs for four 8 $\times$ 8 multipliers and two additions; 2) one DSP for two 8 $\times$ 8 multipliers. In such way, we can fully leverage the DSPs while keeping the ALMs under the constraints.

V. Latency Estimation with Gaussian Process

A. Motivation

Design space exploration (DSE) has been widely used in hardware accelerators [11], [35], [40] for CNNs, to optimize a wide range of hardware parameters in an effort to efficiently map a CNN onto the target FPGA. The DSE process usually involves two approximating models. The models are used instead of running the CNN on real hardware after each hardware iteration with different hardware parameters to collect measurements, which is very time consuming. One model is the resource model, which models the resources for a specific architecture with given hardware parameters in the target FPGA device. The other model is the performance model, which usually estimates the corresponding system performance, e.g. latency, given the chosen hardware and fixed algorithmic properties. Then DSE will try to find the optimal design parameters which achieve the best performance under the resource constraints for a given device.

There are several rather complicated performance estimation frameworks for FPGA-based accelerators [41], [42]. Therefore, practitioners usually resort to an analytic formulation of performance prediction that provides a rough estimate, e.g. for the latency, due to the simplicity of this prediction method. Additionally, the analytic approximation can be easier to integrate into DSE optimisation loop, which is often custom to support a variety of CNNs [5], as in comparison to working with all-round simulation software [43].

Nonetheless by avoiding the use of dedicated simulation software or complicated performance predictors and instead using only an analytic approximation introduces several challenges. First, by formulating an analytic approximation, we usually avoid to count for scheduling which can introduce errors in the prediction. Second, the explicit time to execute a certain operation on hardware varies by on/off-chip communication, synchronisation, control signals, I/O interruptions and in particular for the CNN accelerators - the CNN's architecture, which cannot be covered by analytic estimation. Third, a pure analytic method is unable to account for any collected real-world performance measurements. Therefore, it is necessary to develop a performance estimation method, which provides the user with a reliable guarantee of the expected performance, while not increasing the implementation effort.

B. Our Method

In this work, we propose a novel approach for accurate performance estimation for FPGA-based CNN accelerators, that we used to estimate latency of a given CNN on the accelerator. This method employs a Gaussian process regression (GPR) [44] approach coupled with the standard analytic formulation [11] and the collected measurements.

GPR is a non-parametric, Bayesian approach for regression that can embody prior knowledge/model into the target. It is specified by a mean function $m(\cdot)$ and a covariance function (kernel) $k(\cdot, \cdot)$. The mean function represents the supposed

average of the estimated data. The kernel computes correlations between inputs and it encapsulates the structure of the hypothesised function.

The predictive distribution $p(y_t|\mathbf{X}, \mathbf{y}, \mathbf{X}_t)$ for the targets y_t given the corresponding features $\mathbf{X}_t$ and the training data $(\mathbf{X}, \mathbf{y})$ is defined as a multivariate Gaussian distribution with a predictive mean $\mathbb{E}[y_t|\mathbf{X}, \mathbf{y}, \mathbf{X}_t]$ and a predictive variance $\mathbb{V}[y_t|\mathbf{X}, \mathbf{y}, \mathbf{X}_t]$, which are defined as follows:

$$\begin{aligned} &\mathbb{E}[y_t|\mathbf{X}, \mathbf{y}, \mathbf{X}_t] \\ &= m(\mathbf{X}_t) + k(\mathbf{X}_t, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})^{-1}(y - m(\mathbf{X})) \\ &\mathbb{V}[y_t|\mathbf{X}, \mathbf{y}, \mathbf{X}_t] \\ &= k(\mathbf{X}_t, \mathbf{X}_t) + k(\mathbf{X}_t, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})^{-1}k(\mathbf{X}_t, \mathbf{X})^T \end{aligned} \tag{1}$$

where σ^2 represents the noise amplitude and $\mathbf{I}$ is the identity matrix. The detailed derivations can be found in [45].

In this work, the GP's target is to estimate the latency for a single layer based on the input features. The features include the model's layer parameters introduced in Section III-A and the accelerator's parameters such as the degrees of parallelism (PC and PF), clock frequency and data width. The standard analytic formulation developed in our prior work [11] is used as the mean function of the GP, with the profiling data collected by running the CNN on real hardware as the training data.

The main benefit of using a GP over other methods such as linear regression or gradient tree boosting, that rely on a large number of collected measurements, is that it can use the previously developed analytic formulation, as prior knowledge in a form of $m(\cdot)$. Thus, it reuses any previously developed heuristics and only minimally increases the implementation effort by tuning a small number of hyperparameters, while requiring a smaller number of collected measurements; thanks to the heuristic. Moreover, it can use the previously collected measurements $(\mathbf{X}, \mathbf{y})$ to learn to account for any non-linearities such as on/off-chip communication, synchronisation or control signals.

VI. Evaluation and Experiments

A. Benchmarks

Some typical CNNs have been tested as benchmark models as listed in Table II. These models are widely used for tasks of classification, object detection and segmentation. VGG16 [2] is one of the largest and computationally intensive networks, with serial layer connectivity and uniform kernel size (3x3) across its convolutional layers. ResNet-50 and ResNet-101 [7] represent the mainstream networks that contain the residual connections insides their blocks. Inception-v4 [38] has a more uniform and simplified architecture with concatenative connections compared to ResNet models. SSD [46] has the architecture which builds on VGG16, and a set of auxiliary convolutional layers were added to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer. U-Net [10] is famous for the introduction of large up-sampling (deconvlotional) layers for semantic segmentation and it also has concatenative connections. YOLOv3 [9] is a mainstream network with feature map up-sampling and concatenation. Its feature extractor is built on Darknet-53 which is organized as a series of residual blocks. Therefore, YOLOv3 has all the characteristics we mentioned in Figure 1. It should be noted that besides the mentioned benchmark models, other networks are also supported by our accelerator.

TABLE II
Benchmark Models

Category	Network	Workloads (GOPs)	Characteristic
Classification	VGG16 [2]	30.94	· serial connectivity · uniform kernel size
	ResNet-50 [7]	7.7	· residual connection
	ResNet-101 [7]	15.5	
	Inception-v4 [38]	27.6	· concat. connection
Object Detection	SSD [46]	5.254	· branches
	YOLOv3 [9]	71.4	· up-sampling · concat. connection · residual connection
Segmentation	U-Net [10]	816.9	· up-sampling · concat. connection

B. Implementation Details

Our accelerator was implemented and evaluated on the Intel's Arria 10 device which consists of a high-performance and power-efficient FPGA device, i.e. Arria 10 GX1150 (20 nm), a dual-core ARM Cortex-A9 processor (1.5 GHz) and 2 GB DDR4 memory. The ARM CPU was used to configure the layers' parameters when running each model in our accelerator. All the hardware modules are developed using Verilog HDL. The hardware system was synthesized and placed-and-routed with Quartus Prime Pro 18.1. In the target device, our accelerator achieved the optimal design parameters at PC$\times$PF = 64×64 and the computation engine is run at the clock frequency of 200 MHz.

C. Latency Estimation Results

The evaluation dataset comprises the convolutional layers from three CNNs, i.e. 24 convolutions of SSD [46], 57 convolutions of ResNet-50 [7] and 75 convolutions of YOLOv3 [9]. Each model was executed on the implemented accelerator on Intel Arria GX1150 FPGA. The training of the model was performed on 50% of the data over the three networks. Additionally, for a more comprehensive evaluation leave-one-out cross validation was used, where each time, one sample was left out and all the others were used for training. This process is then repeated for each sample in the dataset. The GPR is implemented using the existing GPflow [47] library and it was trained using an Adam optimiser with the initial learning rate 1×10^{-3} until convergence with respect to the relative error. Matérn 3/2 kernel [45] is chosen as the GP's kernel. The result is shown in Table III in comparison to the standard analytic approximation [11].

The experiment results demonstrate the estimation accuracy improvements provided by the GPR. Compared to the standard method, it reduces the relative error from 27.6% to 8.3%, 33% to 9.2%, 22% to 3.1% for the evaluated models respectively,

TABLE III
LATENCY ESTIMATION WITH GAUSSIAN PROCESS REGRESSION COMPARED TO STANDARD METHOD

	ResNet-50		SSD		YOLOv3	
	standard	GPR	standard	GPR	standard	GPR
Mean absolute error per layer (ms)	0.276	0.112	0.27	0.086	0.408	0.192
Maximum absolute error per layer (ms)	1.07	0.394	1.26	0.323	1.75	0.661
Total estimated latency (ms)	3.67	4.65	2.40	3.24	38.2	47.5
Total reference latency (ms)	5.07		3.57		48.99	
Relative error	27.6%	8.3%	33%	9.2%	22%	3.1%

achieving a maximum of 23.8% and an average of 20.7% reduction in the errors of latency estimation. The results confirm that our method provides a very accurate estimate of latency, and thus accelerates the process of CNN model tuning in order to satisfy the latency requirement for real-time applications. Therefore, the proposed method can largely reduce the design time for the trade-off between accuracy and performance, and improve the hardware design productivity.

D. Resource Efficiency

Table IV shows the resource utilization of the accelerator on Arria 10 GX1150. Owning to the use of 8-bit quantization and the proposed DSP configurations, the low-precision fixed-integer multipliers are implemented individually in soft logic or combined with other multipliers in the DSP blocks, leading to high resource utilization and great compute density. By fully leveraging the DSP blocks, we have a relatively low use of ALMs and registers, which enables our accelerator to work at a relatively high clock frequency (200 Mhz).

TABLE IV
RESOURCE UTILIZATION OF THE ACCELERATOR ON ARRIA 10 GX1150

Resources	ALMs	Registers	DSPs	M20K
Used	303,913	888,576	1,473	2,334
Total	427,200	1,708,800	1,518	2,713
Utilization	71%	52%	97%	86%

Figure 11 shows a breakdown of the resources of each module in the datapath. Since the MM module is the core computation block, it has the highest utilization of ALMs, Registers and DSPs among all the modules. Besides the MM module, the arithmetic operation inside the residual and ReLU blocks are implemented with DSPs that can be configured for element-wise *add* / *subtract* / *multiply* operation. The other two modules, i.e. GAP and Pooling use soft logic to implement the arithmetic operations, and thus they use a relatively high percentage of ALMs and on-chip registers.

E. Compute Efficiency and Model Accuracy

Table V shows a summary of the performance, compute (MAC) efficiency and accuracy for our benchmark models when running on Arria 10 GX1150 device. Our accelerator achieves the throughput of 1.30 $\sim$ 1.59 TOP/s (tera-operations per second), which is up to 97% of the theoretical maximum

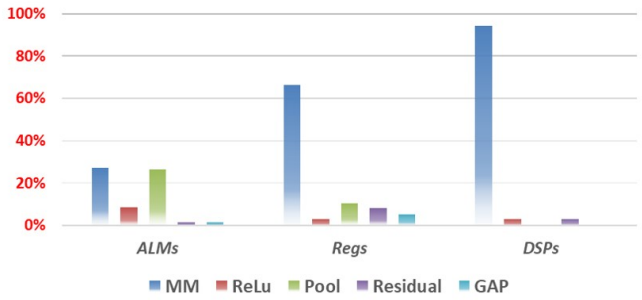

Fig. 11. The resource breakdown of each module in the datapath.

performance. As can be seen, it achieves the compute efficiency of 79.1% $\sim$ 97.0%, depending upon the network. The relative low efficiency of VGG16 is due to the large computation of the first layer since it can only achieve 75% of efficiency as we have discussed in Section IV-B. Nevertheless, our accelerator achieves high compute efficiency of more than 89% for networks with irregular types such as ResNet and YOLOv3. Our framework employs the 8-bit fixed-point quantization scheme in [15], and the resulting accuracy for the CNNs is almost equivalent to that of the original floating point 32-bit (FP32) model, which are within one percentage point of the original FP32 accuracy without retraining.

F. Performance Comparison

Comparison With Embedded GPU: We compare the performance of our design with the widely used high-performance NVIDIA Tegra X1 platform. TX1 has 256 CUDA cores delivering over 1 TOP/s of peak performance with a power consumption of 10W. NVIDIA TensorRT as supplied by the JetPack 3.1 package was run with the NVIDIA cuDNN library and FP16 precision, which enables a highly optimized execution of layers. Although a batched way of processing can fully utilize the parallelism of GPU on TX1, it is not a good choice for real-time processing because it increases latency, as discussed in Section I. Therefore, on all evaluated platforms, the benchmarks are run with a batch size of 1.

Performance comparison is shown in Table VI. As we can see, the GPU performance has a large divergence across the evaluated models from 131 $\sim$ 322 GOP/s compared to ours of 1.3 $\sim$ 1.59 TOP/s on FPGA. In general, GPU performs better on larger CNN models with regular shape and serial connectivity such as in cases of VGG16 and U-Net. However, the GPU's performance decreases dramatically on smaller models or models with residual or concatenative connections. As a result, GPU TX1 has the lowest performance of 131 GOP/s for YOLOv3 among all the evaluated models. Owning to our customized and careful design for the irregular connections, our accelerator achieves an overall high compute efficiency across all benchmark models. The proposed accelerator achieves 4$\times$ $\sim$ 10.5$\times$ speedup in terms of the throughput of GOP/s. Even for VGG16 and U-Net, we achieve similar energy efficiency of GOP/s/W compared to GPU. Apart from these two models, we achieve 1.32$\times$ $\sim$ 2.33$\times$ improvements on the energy efficiency compared to GPU.

Comparison With Previous FPGA Accelerators: Table VII gives the performance comparison of our design against prior FPGA-based accelerators. All results are based on the batch size equal to 1, and the energy efficiency results are normalized using the board power consumption for fair comparison across all works. The total number of DSPs in device is used to compare the resource efficiency (GOP/s/DSP) because the utilization of DSPs can be regarded as a metric of the quality of the hardware architecture design of FPGA-based accelerators.

For all evaluated networks, our accelerator outperforms all other accelerators in terms of both resource efficiency (GOP/s/DSP) and energy efficiency (GOP/s/W), as shown in Table VII. Among all the accelerators, we achieve the best resource efficiency of 1.0 GOP/s/DSP and energy efficiency of 33.8 GOP/s/W. In [31], the authors achieved a similar energy efficiency of 30.7 GOP/s/W to our work. However, their work only implemented AlexNet which has uniform and regular shape and its performance will be impacted negatively with other CNN topology with irregular connections. Besides, [31] used a batch size of 1 for convolution layers, and 96 for the fully connected layers, which increased the throughput but actually also increased the latency. Compared to the state-of-art implementation of CNNs with irregular shapes in [35], we achieve a resource efficiency improvement of 6.13× and an energy efficiency improvement of 2.2× for VGG16 and ResNet.

VII. Conclusion

This paper presents an accelerating framework towards the full-stack acceleration of CNNs on FPGAs. Computational functions such as convolutional, deconvolutional and full-connected layers are mapped in a unified architecture by exploiting different level of parallelism and fully leveraging the DSPs. Besides, the proposed accelerator addresses the efficiency of the irregular connections in CNN models such as residual and concat connections by the smart cache design. Quantitative evaluation results demonstrate that our accelerator outperforms the performance density and energy efficiency of existing state-of-the-art FPGA-based accelerators, achieves a high compute efficiency, and therefore provides a highly-optimized, specialized hardware accelerator for machine learning acceleration.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, 2016, pp. 379–387.

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. pattern analysis and machine intelligence.*, vol. 39, no. 12, pp. 2481–2495, 2017.

[5] S. Liu and W. Luk, "Towards an Efficient Accelerator for DNN-Based Remote Sensing Image Segmentation on FPGAs," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 187–193.

[6] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 1–14.

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

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *In Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[11] S. Liu, H. Fan, X. Niu, H.-C. Ng, Y. Chu, and W. Luk, "Optimizing CNN-based Segmentation with Deeply Customized Convolutional and Deconvolutional Architectures on FPGA," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 19:1–19:22, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3242900

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

TABLE V
PERFORMANCE AND ACCURACY ON BENCHMARK MODELS

	Dataset	Network	Latency[a](ms)	Throughout[a] GOP/s	Energy Efficiency[b] GOP/s/W	MAC Efficiency	Accuracy			
							standard	This Work	FP32	diff.
Classification	Imagenet [36]	VGG16	23.9	1295	28.8	79.1%	top-1	70.23%	70.98%	-0.75%
		ResNet-50	5.07	1519	33.8	92.7%		74.73%	75.13%	-0.4%
		ResNet-101	9.79	1590	35.4	97.0%		76.72%	76.47%	+0.25%
		Inception-v4	21	1314	29.2	80.2%		80.06%	80.1%	-0.04%
Object Detection	FDDB [48]	SSD	3.57	1472	32.7	89.8%	IoU≥0.5	83.5%	83.5%	0
	COCO [49]	YOLOv3	48.99	1457	32.4	89.0%	IoU≥0.5	56.6%	57%	-0.4%
Segmentation	Cityscapes [50]	U-Net	543.19	1504	33.4	91.8%	Pixel	95.36%	95.53%	-0.17%
							Class	93.68%	93.75%	-0.07%
							IoU	85.96%	86.55%	-0.59%

[a] The batch size is set to 1.
[b] The energy efficiency is quoted in in giga-operations per second per watt (GOP/s/W) and the total board power consumption (45W) is used for computation.

TABLE VI
COMPARISON WITH EMBEDDED GPU TX1

	GPU TX1 (fp16)		FPGA GX1150		
	Latency/ms	GOP/s	Latency/ms	Speedup	
				GOP/s	GOP/s/W
VGG16	96.5	318	23.9	4.04×	0.9×
ResNet-50	53.09	145	5.07	10.5×	2.33×
ResNet-101	84.52	183	9.79	8.63×	1.92×
Inception-v4	158.00	174	21	7.52×	1.67×
SSD	21.22	247.5	3.57	5.94×	1.32×
YOLOv3	454	131	48.99	9.27×	2.1×
U-Net	2540	322	543.19	4.68×	1.04×

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[15] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018, pp. 2704–2713.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[17] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[18] B. McDanel, S. Q. Zhang, H. Kung, and X. Dong, "Full-stack Optimization for Accelerating CNNs with FPGA Validation," *arXiv preprint arXiv:1905.00462*, 2019.

[19] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *FCCM*, 2017, pp. 101–108.

[20] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *FPGA*, 2017, pp. 35–44.

[21] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[23] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[24] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[25] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021736

[26] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

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

[29] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[30] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 57–64.

[31] U. Aydonat *et al.*, "An OpenCL™Deep Learning Accelerator on Arria 10," in *FPGA*, 2017, pp. 55–64.

[32] A. Yazdanbakhsh *et al.*, "FlexiGAN: An End-to-End Solution for FPGA Acceleration of Generative Adversarial Networks," in *FCCM*, 2018.

[33] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "Gna: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2519–2529, Nov 2018.

[34] S. Liu, C. Zeng, H. Fan, H.-C. Ng, J. Meng, and W. Luk, "Memory-Efficient Architecture for Accelerating Generative Networks on FPGAs," in *IEEE International Conference on Field Programmable Technology (FPT)*, 2018.

[35] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: mapping regular and irregular convolutional neural networks on FPGAs," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.

[37] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing CNN-Based Object Detection Algorithms on Embedded FPGA Platforms," in *ARC*. Springer, 2017, pp. 255–267.

[38] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[39] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[40] R. Zhao, H.-C. Ng, W. Luk, and X. Niu, "Towards efficient convolutional neural network for domain-specific applications on FPGA," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 147–1477.

[41] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 129–132.

[42] R. Yasudo, J. Coutinho, A. Varbanescu, W. Luk, H. Amano, and T. Becker, "Performance estimation for exascale reconfigurable dataflow platforms," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 314–317.

[43] "Modelsim®." [Online]. Available: https://www.mentor.com/products/fv/modelsim/

[44] C. K. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in neural information processing systems*, 1996, pp. 514–520.

[45] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[46] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[47] A. G. De G. Matthews, M. Van Der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman, "Gpflow: A gaussian process library using tensorflow," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1299–1304, 2017.

[48] V. Jain and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," UMass Amherst technical report, Tech. Rep., 2010.

[49] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[50] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[51] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, Jan 2018.

PLACE PHOTO HERE

Shuanglong Liu received the B.Sc. and M.Sc. degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, and D.Phil. degree in Electric Engineering from Imperial College London, London, U.K. He is currently a Research Associate with the Department of Computing, Imperial College London, London, U.K. He has published over 20 research papers in peer-referred journals and international conferences. His current research interests include reconfigurable and high performance computing for Convolutional Neural Networks (CNNs) and statistical inference problems.

PLACE PHOTO HERE

Hongxiang Fan received the B.S. degree in electronic engineering from Tianjin University, Tianjin, China, in 2017, and the master's degree from the Department of Computing, Imperial College London, London, U.K., in 2018. He is currently a Ph.D. student in Machine Learning and High-Performance Computing at Imperial College London. His current research focuses on efficient algorithm and acceleration for Machine Learning applications.

PLACE PHOTO HERE

Martin Ferianc received the MEng. degree from the Department of Electronic Engineering, Imperial College London, London, U.K., in 2019. He is currently a Ph.D. student at University College London. His current research interests include convolutional neural networks and neural architecture search applied to computer vision tasks.

PLACE PHOTO HERE

Xinyu Niu is the Co-Founder and CEO of Corerain Technologies in Shenzhen, China. He received the B.Sc. Degree from Fudan University, Shanghai, China, and the M.Sc. and Ph.D. degrees in computing science from Imperial College London, London, U.K. His current research interests include developing applications and tools for reconfigurable computing that involves runtime reconfiguration.

PLACE PHOTO HERE

Huifeng Shi has been the executive deputy director in the State Key Laboratory of Space-Ground Integrated Information Technology (SGIIT) in Beijing, China since 2017. He received the B.Sc. Degree from Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China and the M.Sc. Degree from National University of Defense Technology, Changsha, China. His research interests include image processing, multi-sensor information fusion and data mining.

TABLE VII
COMPARISON WITH PREVIOUS FPGA ACCELERATORS

	Ma *et al.* [25] in FPGA 2017	**Aydonat *et al.* [31] in FPGA 2017**	**Guo *et al.* [51] in TCAD 2018**	**Liu *et al.* [11] in TRETS 2018**	**Venieris *et al.* [35] in TNNLS 2019**		**This Work**		
Platform	Intel GX1150	Intel GX1150	Xilinx Zynq-7020	Xilinx Zynq-7045	Xilinx Zynq-7045		Intel GX1150		
Frequency (MHz)	150	303	214	200	125		200		
Bit-width	8-16 bit fixed	16-bit float	8-bit fixed	16-bit fixed	16-bit fixed		8-bit fixed		
#DSP	1518	1518	220	900	900		1518		
Logic (ALMs/LUTs)	427K	427K	53K	218K	218K		427K		
Power[a](W)	45	45	3.5	9.6	9.6		32		
CNN Model	VGG16	AlexNet	VGG16	Optimized U-Net	VGG16	ResNet-152	VGG16	ResNet-50	U-Net
Latency[b](ms)	47.97	not reported	364	58.0	249.5	156.4	23.9	5.07	543.19
Performance (GOP/s)	645.25	1382	84.3	107	124	147	1295	1519	1504
Resource Efficiency[c] (GOP/s/DSP)	0.425	0.91	0.38	0.12	0.14	0.163	0.86	1.0	0.99
Energy Efficiency[d] (GOP/s/W)	14.3	30.7	24.1	11.2	12.9	15.3	28.8	33.8	33.4

[a] In all works, the power consumptions are tested using the board power.
[b] All works use the batch size of 1.
[c] The resource efficiency results are evaluated using the total DSPs in the devices for all designs.
[d] The energy efficiency results are normalized here using the board power for all works, while [35] originally used the power consumption with subtracted idle power.

PLACE
PHOTO
HERE

Wayne Luk received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from Oxford University, Oxford, U.K. He founded and leads the Computer Systems Section and the Custom Computing Group, Department of Computing at Imperial College London. He was a Visiting Professor at Stanford University, Stanford, CA, USA, and Queens University Belfast, Belfast, U.K. He is currently a Professor of Computer Engineering at the Imperial College London, London, U.K. He has been an author or editor for six books and four special journal issues. Dr. Luk is a member of the Program Committee of many international conferences, such as the IEEE International Symposium on Field-Programmable Custom Computing Machines, the International Conference on Field-Programmable Logic and Application (FPL), and the International Conference on Field-Programmable Technology (FPT). He is a Fellow of the Royal Academy of Engineering and the British Computer Society Ltd. He had 15 papers that received awards from various conferences, such as the IEEE International Conference on Application-specific Systems, Architectures and Processors, FPL, FPT, the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, X Southern Programmable Logic Conference, and the European Regional Science Association. He also received the Research Excellence Award from the Imperial College London in 2006. He was a founding Editor-in-Chief of the ACM Transactions on Reconfigurable Technology and Systems