Doctoral Dissertation

A Fully-Pipelined Inference Accelerator for Deep Convolutional Neural Networks

NGUYEN VAN CAM
Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Professor Yasuhiko Nakashima
Computing Architecture Lab. (Division of Information Science)

June 02, 2023

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

NGUYEN VAN CAM

Thesis Committee:

Supervisor	Prof. Yasuhiko Nakashima
	(Professor, Division of Information Science)
Co-supervisor	Prof. Yuichi Hayashi
	(Professor, Division of Information Science)
	Assoc. Prof. Renyuan Zhang
	(Associate Professor, Division of Information Science)
	Asst. Prof. Kan Yirong
	(Assistant Professor, Division of Information Science)

A Fully-Pipelined Inference Accelerator for Deep Convolutional Neural Networks*

NGUYEN VAN CAM

Abstract

Due to the high speed and power efficiency of the field-programmable gate array (FPGA), many FPGA-based inference accelerators for deep convolutional neural network (CNN) have been widely adopted. Large-scale CNNs require intensive computations as well as a large amount of storage space and memory access. However, low bandwidth off-chip memories are a main challenge for data transmission between external memory and FPGA-based CNN inference accelerator.

In this research, we develop the following to improve performance and power efficiency. First, we use a high bandwidth memory (HBM) to expand the bandwidth of data transmission between the off-chip memory and the accelerator. Second, a fully-pipelined manner, which consists of pipelined inter-layer computation and a pipelined computation engine, is implemented to decrease idle time among layers. Third, a multi-core architecture with shared-dual buffers is designed to reduce off-chip memory access and maximize the throughput.

We designed the proposed accelerator on the Xilinx Alveo U280 platform with in-depth Verilog HDL instead of high-level synthesis as in the previous works and explored the VGG-16 model to verify the system during our experiment. With a similar accelerator architecture, the experimental results demonstrate that the memory bandwidth of HBM is 13.2× better than DDR4. Compared with other accelerators in terms of throughput, our accelerator is 1.9×/1.65×/11.9× better than FPGA+HBM2 based / low batch size GPGPU / low batch size

*Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, June 02, 2023.

CPU. Compared with the previous DDR+FPGA / DDR+GPGPU / DDR+CPU based accelerators in terms of power efficiency, our proposed system provides 1.4-1.7×/1.7-12.6×/6.6-37.1× improvement with the large-scale CNN model.

Keywords:

Convolutional Neural Network (CNN), Deep Neural Network (DNN), Image Classification, FPGA, High Bandwidth Memory (HBM), Accelerator, Power Efficiency.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Nowadays, research on Deep Neural Networks (DNN) is showing a huge improvement over traditional algorithms in machine learning [1, 2, 3, 4]. Various network models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have been proposed for image, video, and speech processes. CNN improved the image classification accuracy [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and further improved object detection [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27] with its superior feature extraction. RNN achieves high performance in extracting time-continuous features such as speech recognition [28, 29, 30, 31, 32], video captioning [33, 34, 35, 36, 37], etc. In general, DNN is highly suitable for a wide range of pattern recognition problems. This ability makes DNN a promising candidate for many artificial intelligence applications.

Table 1. Top-1 Accuracy and Resource Utilization of State-of-the-Art Convolutional Neural Network Models.

Metrics	**AlexNet**	**ResNet152**	**MobileNet**	**ShuffleNet**
Year	2012	2016	2017	2017
#Param	60M	57M	4.2M	2.36M
#Operation	1.4G	22.6G	1.1G	0.27G
Top-1 Accuracy	61%	79.3%	70.6%	67.6%

M: Million ($\times 10^6$), **G**:Giga ($\times 10^9$).

Recent deep convolutional neural networks (CNNs) have became popularly used due to their superior efficiency in computer vision fields [38, 39, 40, 41, 42, 43, 44], such as image classification, object recognition, pedestrian detection, object tracking, etc. CNNs are computationally intensive models that achieve high accuracy on large datasets [45, 46, 47, 48, 49]. However, they require a huge amount of computational cost, storage space, and high bandwidth memory access. For example, the CNN models from Table 1 [50] show that the ResNet152 model has high performance but the model is very complicated instead. Specifically, the

number of learned parameters is up to 57 million parameters. If each parameter is 32-bit then it takes at least 218MB of memory to store them. In addition, the number of calculations is up to 22.6 billion. Latest models such as MobileNet or ShuffleNet have significantly reduced number of learned parameters and number of calculations compared to ResNet152 model, but obviously the accuracy will decrease. The balance between accuracy with storage space, computational cost, and power consumption for CNN models on platforms is still a critical problem today.

Central processing unit (CPU) and general purpose graphic processing unit (GPGPU) are two well-known general-purpose hardware platforms. GPGPUs can achieve performance up to 10 TOPS, while CPU can only achieve 10- 100 GFLOPS [50]. GPGPUs that specialize in parallel processing are called "Single Instruction Multi-Data (SIMD)" [51, 52]. GPGPUs are more suitable than CPUs for large amounts of training data, matrices, and complex operations in deep learning. Therefore, GPGPUs with massively parallel computational capacity are widely used for large-scale, high-cost CNN models. Despite producing high performance for CNN implementation, GPGPUs still suffer from huge power consumption requirements, resulting in poor power efficiency. Particularly, embedded devices and mobile devices are not possible to use GPGPU to process the large-scale data efficiently.

To improve power efficiency, field programmable gate array (FPGA) platform, which can achieve high performance and low power cost, is a good candidate for replacing GPGPU to improve power efficiency[53, 54, 55, 56, 57]. More specially, FPGA-based accelerators have greater reconfigurability and shortened deployment time compared to application specific integrated circuit (ASIC) [58, 59, 60]. The performance of current FPGA platforms can achieve, for example, 9.2 TFLOPS by Intel Stratix 10 FPGA [61] or even 40 TFLOPS by Intel Agilex FPGA platform [62].

Many existing FPGA-based CNN inference implementations with low-bandwidth computation [63, 64, 65, 66, 67, 68], compression mechanisms [69, 70, 71] or quantization [72, 73, 74, 75, 76, 77] have been applied for low numerical precision to reduce the external memory bandwidth pressure. In addition, using on-chip buffers (e.g., [78, 79, 80]) is a popular solution for limiting bandwidth memory.

However, CNN models are becoming larger for analyzing and extracting features from input images , requiring more arithmetic operations and storage space. The existing FPGA-based CNN inference accelerators with off-chip memory (DDR) could not satisfy the memory bandwidth requirement for computation with large-scale CNN model due to FPGA resources constraints.

The release of high bandwidth memory (HBM) combined with the accelerator on FPGA promises to create highly efficient CNN inference accelerators. From the specification of the Xilinx Alveo U280 platform, the bandwidth memory of two HBM2 (second-generation of high bandwidth memory) DRAM stacks can achieve 460 GB/s at 900 MHz clock rate, which is more than 12× the performance of DDR4 memory bandwidth (only 38 GB/s [81]). Moreover, many evaluations have been conducted to demonstrate the experimental performance of HBM2. For instance, Huang et al. [82] validated the much higher memory throughput of HBM2 than DDR4 (425 GB/s vs. 36 GB/s). Moreover, Samsung HBM2 throughput can achieve 3× improvement and save 80% power consumption compared to the GDDR-based system [83]. Based on the above advantages, the combination of FPGA-based accelerators and HBM2 promises to improve performance but also maintain power efficiency compared to the GPGPU platform.

1.2 Research Contribution

This study's main contributions can be summarized as follows:

- We implement a CNN inference accelerator on the FPGA and HBM2 platform at the system-on-chip level with in-depth Verilog HDL instead of high-level synthesis as the previous researchs. The real memory bandwidth of HBM2 is 13.2× larger than DDR4's bandwidth.

- In order to increase the accelerator's performance, we adopt a fully-pipeline mechanism, which contains inter-layer pipeline and a pipelined architecture inside of computation engine.

- The proposed accelerator consists of a multi-core architecture with dual buffers for storing the feature maps and shared-dual buffers for saving kernel parameters to reduce the off-chip memory access and improve the throughput.

- We implement the small/large scale VGG-16 model on CIFAR-100/ImageNet dataset to evaluate the accelerator. Then, we closely analyze the effect of HBM2 on the accelerators by making a fair comparison to the HBM2 based and DDR memory based. Since the existing FPGA-based CNN inference accelerators have poor performance and low flexibility, the proposed system is compared to other high-performance and high-flexibility hardware platforms such as DDR+CPU/DDR+GPGPU with various batch sizes in terms of throughput and power efficiency.

- The throughput of the proposed system reaches 912.7 GOP/s and 22.48 GOP/s/W in peak throughput and power efficiency, which surpasses NVIDIA GeForce RTX3090 GPGPU, Intel i9-10940 CPU, and FPGA-based CNN inference accelerators [84, 85, 86] .

1.3 Dissertation Layout

The thesis is divided into five chapters which are organized as follows:

- Chapter 1 introduces the overview, contributions, and layout of this research.

- Chapter 2 gives an overview of the deep neural network. Then, the preliminary of convolution neural networks is summarized. Finally, the details of structure, features and specifications of High Bandwidth Memory (HBM) on Xilinx Alveo U280 board are presented.

- Chapter 3 analysis the related works and their drawbacks.

- Chapter 4 shows the details of the system architecture. In this chapter, the structure of the system , accelerator and their workflow are presented. Moreover, the system experiment setup is also showed in detail.

- Chapter 5 analysis the results. This chapter also evaluates by comparing with related works and CNN inference implementation on DDR-based CPU/GPGPU.

2 Background and Related Work

This chapter firstly introduces the background of Artificial Intelligence history. Then, the overview of Artificial Neural Networks (ANNs), Deep Neural Networks (DNNs), and Convolutional Neural Networks (CNNs) is summarised. Next, I present an overview of the structure and features of High Band Memory (HBM). Finally, the drawbacks of CNN inference accelerators based on the Central Processing Unit (CPU), General Purpose Graphic Processing Unit (GPGPU), and Field Programmable Gate Array (FPGA) are analyzed in detail and the solutions are also given.

2.1 Overview of Artificial Intelligence

Artificial intelligence (or AI) can be defined as "the study of making computers that do things that humans need intelligence to do" [87]. This expanded definition includes not only the first human thought processes, but also technologies that help computers achieve intelligent tasks even if they do not necessarily simulate thought processes human thought. AI is a field of research that synthesizes and analyzes computational agents that operate intelligently. Actors are activities in an environment to perform a certain task. In reality, the agents can be worms, airplanes, people, robots, companies, etc.

The core goal of AI is to learn the principles that make intelligent behavior possible in natural or artificial systems. This is done by:

- Analysis of natural and man-made agents.
- Construct and test hypotheses about what it takes to build intelligent agents.
- Design, build, and test computational systems that perform common tasks.

Artificial intelligence develops with the aim of replacing human intelligence with machines.

2.1.1 History of AI

AI technology has been interesting and engaging since its inception. For decades, it has been one of the key technologies for robotics. From the development of this technology, we will see that we are moving towards advanced mobile intelligence and getting closer and closer. The field of artificial intelligence dates back to the 1940s. World War II and the need for rapid technological advancement to combat the enemy prompted the creation of the field thanks to mathematician Alan Turing and neuroscientist Grey Walter [87].

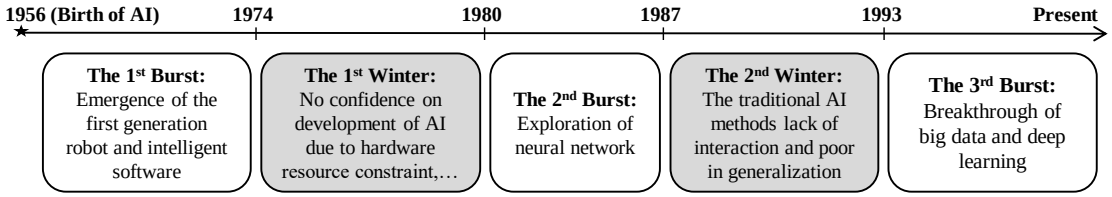

Figure 1. Overview of AI history.

In 1956, the famous Dartmouth Conference proposed the claim: "It is possible to describe exactly any aspect of learning or any other feature of intelligence so that a machine can be built to emulate it" [88]. Therefore, this conference has been recognized as the birth of AI. Over the past 60 years, AIs have gone through various stages based on their specific development features, as shown in Fig. 1 [89]. In addition, because of certain criticisms and limitations, AI also experienced two winter periods (freezing, almost no significant development). However, in all these years, researchers never stop their work in related techniques, and breakthroughs of various applications are also accompanied.

The first phase, named the Inference Age (1956-1974), is considered the golden age for AI development. The most important task in this stage is to make computers capable of logical reasoning. With the emerging smart techniques, people have gone through many "unbelievable" changes. For example, robots could both look and behave like humans, or machines could learn to speak and communicate with people [90]. At that time, all researchers were optimistic that AI can be easily achieved by logical reasoning.

However, AI failed to keep its growth in the 1970s. The optimism and confidence of AI researchers set extremely high expectations and caused them to

overlook real difficulties. economic problems they faced, such as hardware resource constraints, etc. As a result, financial support was eventually cut off for scalar research into AI. By 1974, almost no funding for AI projects could be found. The first "winter" (freeze) in AI development officially begins.

As the 1980s entered, knowledge engineering (perceptron) became the key word of AI research, the second phase of the re-evolution of AI (1980 - 1987). Significant technological development during this period must have been a combination of multiperceptron and back-propagation methods. During this period, the emphasis of AI research was changed from laboratory research to practical applications.

With the commercialization of AI techniques, there are more and more specialized systems, natural language processing systems, and more. engaged in this field. Accordingly, it has achieved great economic and social benefits, and demonstrated the broad prospects of AI applications. However, in the late 1980s, after more than a decade of prosperity and significant progress in several fields, AI research began to emerge in crisis once again. In general, there are mainly two problems: one is the so-called interaction problem, which is the traditional AI method that is difficult to interact with the environment. The second is poor generalization, that is, traditional AI methods are only suitable for certain specialized systems, difficult to scale to larger and broader complex systems.

After the second "freeze period", AI has begun a new round of discovery since the early 1990s (1993 - present), when it was successfully applied across the technology industry. The impressive point of this period was that computers should have a certain ability to learn on their own with or without human help [89]. To this end, massive data with rich information is indispensable, and the core task is to analyze potential features and patterns embedded in unstructured big data from multiple sources. This task has brought a new challenge to traditional AI methods. Instead of conventional research, it is divided into competitive subfields that focus on specific problems or approaches. Deep Learning is a good example and the most important one is the Deep Belief Network (DBN) proposed by Geoff Hinton et. al. in 2006, which demonstrated how a multi-level neural network can be efficiently pre-trained with large amounts of known data. After several years of development, deep learning was successfully applied in image

recognition producing more accurate results than human candidates in 2011. The achievements mentioned above are inseparable. Now is a new era for AI research, and it will certainly bring more benefits and promote the development of mobile intelligence.

2.1.2 AI Research Approaches

In the history of development, scientists divided into 4 main problems to solve about AI: acting humanly, thinking humanly, thinking rationally, acting rationally.

In which, the level of computer simulation like a human is the most difficult, and this is also the goal that scientists are aiming for. In addition, AI also focuses on the ability of computers to reason, including the following characteristics:

- Reasoning: the ability to solve problems using logical reasoning.
- Knowledge: the ability to represent knowledge about the surrounding world (understanding how many objects and situations exist in the real world, being able to classify those objects/events).
- Planning: the ability to set and achieve stated goals based on demonstrated knowledge.
- Communication: the ability to understand human written and spoken language.
- Perception: the ability to reason about the world from visual images, sounds, and other sensory inputs.

There are two approaches when researching or developing AI-related problems:

- Rules-based techniques: this was the approach in the early days when the field of AI was studied. This technique is based on pre-fixed rules (knowing possible cases in advance and the behaviors corresponding to that case), they work on the principle of "if - then", "opposite" again - then" (if-then-else). The limitation of this technique is the limit of possible scenarios. Instances that are newly spawned during the operation will not be processed.

- Machine learning techniques. This technique well addresses the limitation of the rules-based approach. Possible cases may or may not be present, newly arising cases are handled based on the characteristics of the data set that the system has learned.

Since the 1950s, Machine Learning (ML), a subset of Artificial Intelligence (AI), has revolutionized several fields over the past few decades. Neural Network (NN) is a subfield of ML and it is this subfield that gives rise to Deep Learning (DL). Since its inception, DL has created more disruption than ever, showing outstanding success in almost every application area. Fig. 2 shows the relationship between deep learning (bottom) and artificial intelligence (top) [91]. Deep learning (Deep-Learning, DL), which uses the deep learning or hierarchical learning architecture, is an ML class that was largely developed from 2006 on. Learning is a process that involves estimating model parameters so that the learned model (algorithm) can perform a particular task. For example, in an Artificial Neural Network (Artificial NeurAl Network, ANN), the parameters are the weight matrix. DL, on the other hand, consists of several layers in between the input and output layers, allowing multiple stages of hierarchical non-linear information processing units to be presented to exploit the learning and sample classification [92, 93] . The learning method based on data representation can also be defined as representation learning [94]. Recent literature states that DL-based representation learning consists of a system of features or concepts in which high-level concepts can be defined from low-level and low-level concepts can be defined. defined from high levels. Several studies have shown that DL has been described as a universal learning method capable of solving almost all kinds of problems in various application domains. In other words, DL is not a solution to solve a particular problem [95].

This approach is divided into subgroups based on the learning method:

Supervised Learning - SL: Is an approach of Machine Learning to make computers capable of learning. It is a method that uses previously labeled data to infer the relationship between input and output. These data are called training data and they are input - output pairs. Supervised learning will look at these training sets so that it can make an output prediction for a new input that has never been encountered. For example house price prediction, email classification.

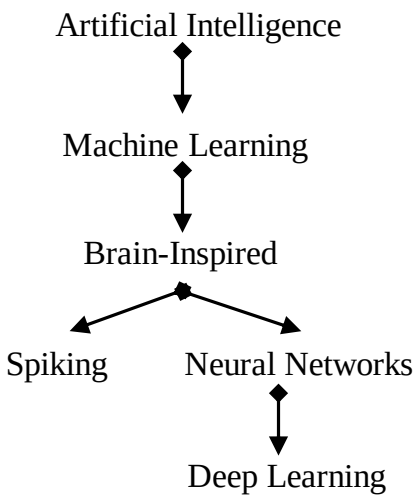

Figure 2. Deep Learning in the context of Artificial Intelligence.

In SL, observables are required to be labeled first. This is one of the disadvantages of this method, because it is not always easy to correctly label observations. For example, in translation, a sentence of the original language can be translated into many different versions in the language to be translated. However, labeled observations is also the advantage of SL because once a large dataset has been correctly labeled, training becomes much easier than when unlabelled data.

Unsupervised Learning - UL: Unsupervised learning uses previously unlabeled data to make inferences. We don't know the output data or the label, only the input data. The UL algorithm relies on the structure of the data to do something, such as grouping or reducing the dimensions of the data to facilitate storage and computation. This method is often used to find the structure of a data set. However, there is no method of evaluating the structure to find out whether it is true or false. For example, clustering data, extracting the main component of a certain substance.

Reinforcement Learning - RL: The RL method focuses on making it possible for an agent in the environment to act in such a way as to obtain as much reward as possible. Unlike SL, it does not have a pre-labeled data pair as input and also does not evaluate actions as true or false. It is the problem that helps a system automatically determine the behavior based on the situation to achieve

the maximum benefit (maximizing the performance). Currently, RL is mainly applied to Game Theory, algorithms need to determine the next move to achieve the highest score.

2.1.3 Some applications of AI

AI has wide applications in science and manufacturing industries, especially in the field of analysis and processing of huge volumes of data (Big Data). Some applications such as:

- Natural Language Processing (NLP) [96, 97]: word processing, human-machine communication, etc.
- Recognition (Pattern Recognition) [98, 99]: speech recognition, handwriting, fingerprint, computer vision (Computer Vision), image processing, etc.
- Search Engine [100, 101].
- Medical diagnostics [102, 103, 104]: X-ray image analysis, automated diagnostic systems.
- Bioinformatics [105, 106]: Gene sequence classification, Gene/Protein formation process.
- Physics [107, 108]: analyzing astrophotography, interactions between particles, etc.
- Economics - finance [109, 110]: detecting financial fraud, credit card fraud, analyzing the stock market, etc.
- Game [111, 112]: automatically play chess, decide the actions of virtual characters.
- Robots [113, 114]: support or act on behalf of humans in toxic environments, etc.

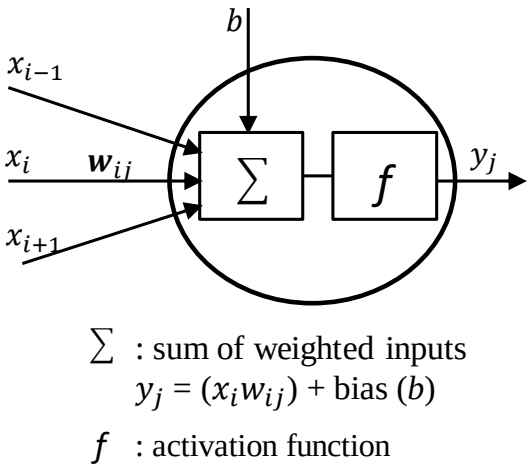

Figure 3. A structure of a neuron with the fixed weight and bias.

2.2 Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs)

An Artificial Neural Network (ANN) is a computational model. They are formed by mimicking the structure and function of the human brain, consisting of billions of neurons and synapses [115, 116]. A synapse is a connection between nodes, or neurons Fig. 3, in an artificial neural network. Similar to biological brains, the connection is controlled by the strength or amplitude of a connection between both nodes, also called the synaptic weight. Multiple synapses can connect the same neurons, with each synapse having a different of influence (activation) on whether the neuron is "activated" and activate the next neuron or not. An Artificial Neural Network is first trained using an available data set (traning phase), then the trained model will be used to predict other input data patterns (inference phase).

Referring to Fig. 3, y is the output of the neuron and is activated by the activation function, which determines whether the neuron is activated or not. For this given neuron, x is the vector of the input variables, w is the weight vector of the edges, and b is the constant bias [117].

Deep Learning is a complex neural network containing many hidden layers between the input layer and the output layer Fig. 4. The layers are made up of

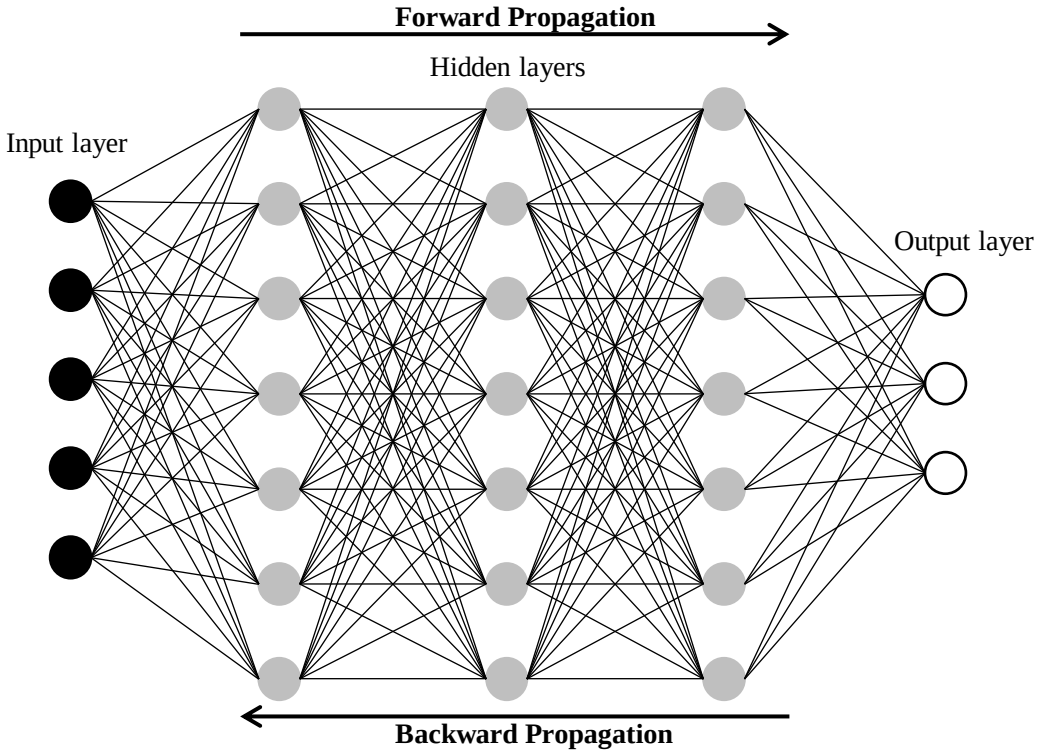

Figure 4. An example of structure of Deep Neural Network (DNN). A DNN structure consists of three types of layers: Input layer, hidden layer and output layer. The DNN performs both forward-propagation and back-propagation during the training phase, and only forward-propagation during the inference phase.

the neurons described above. The number of hidden layers and the number of neurons at each layer (shape and size) depends on the complexity of the input data (application) and the designer's goals. DNN always has a trade-off between high performance and network complexity (many layers, large number of neurons leads to high computational cost and latency). The input of a DNN is a set of values representing the characteristics of the input data (which can be pixels of the input image, waveform magnitude values, sound, values representing the state of a system or a game, etc). From those input data, DNN conducts the process of extracting features and predicting what the meaning of the input data is.

Machine learning works in two main phases: training and inference. To build a model for a certain application, the data set for that application must be divided

into at least two parts: one for the training phase, the other for the inference phase.

- Training phase. The network "learns" to form the characteristics of the network for a given application by performing both forward-propagation and back-propagation processes. The forward-propagation is the process of extracting features of input data based on learned parameters to predict output results. The back-propagation is the process of calculating errors and adjusting model parameters. Both these processes are performed repeatedly (epoch) to get the optimal parameters for the model.

- Inference phase. The network only performs the forward-propagation at this phase. To predict the output results, the optimal parameters, which have been learned from the training phase and the data in the inference dataset, are used.

2.3 Preliminary Convolutional Neural Networks

Convolutional Neural Network (CNN) is a common form of DNN. CNN is a combination of typical layers: convolutional layer (*conv*), pooling layer (*pool*), and fully connected layer (*fc*). Each layer generates a higher level abstraction of the corresponding input data, called feature maps. Currently, there are many types of CNNs proposed by employing a very deep hierarchy of layers. They are able to achieve superior performance in many applications such as image understanding [118, 119, 120], speech recognition [121, 122, 123], robotics [124, 125, 126, 127]. This research only focuses on the task of image classification. Fig. 5 is an example of an image classification application using CNN, which extracts features from the input image and outputs the probability of the corresponding class.

CNN inference indicates the CNN feed-forward propagation of a batch of input images. In this section, the basic computation of each layer in CNN is also presented in detail and visualized in Fig. 6. Table 3 gives the shape of tensors (multi-dimensional matrices) and Fig. 6 [128] shows formulas of typical layers in CNN, where $\{d, v, u\} \in \{[1:D], [1:V], [1:U]\}$, respectively.

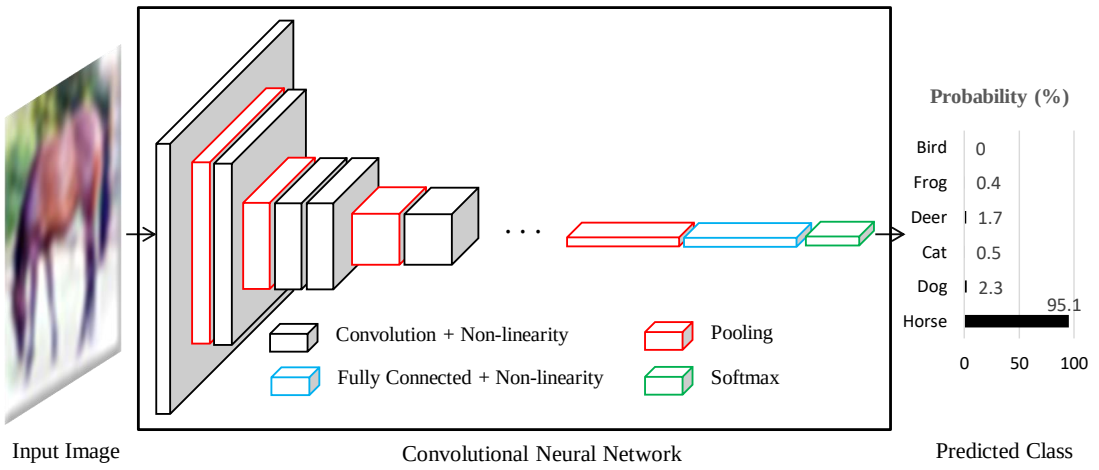

Figure 5. Architecture of the Convolutional Neural Network (CNN) in image classification application.

Convolutional layers: A convolutional layer extracts the feature (output feature maps) from the input feature maps as illustrated in Fig. 6(a), (b) and (c)

Table 2. Dimensions of tensors.

Variable	Description
$\mathbf{X}$	Input Feature Maps (IFMs)
$\mathbf{Y}$	Output Feature Maps (OFMs)
Θ	Learned weight tensor
β	Learned bias tensor
B	Batch size
R	Height of square *pool* window
W/H/C	Width / Height / Depth of IFMs
U/V/D	Width / Height / Depth of OFMs
K/J	Width / Height of kernel window

and the computation in Eq. 1. Each element of the 3-D *conv* layer output feature map is the result of element-wise multiplication and accumulation between the 3-D input feature map (or the input image with the first *conv* layer) and the learned 4-D weight kernel and added by the learned bias kernel. Eq. 1 shows how to calculate each output element of the *conv* layer (OFM), where $\{d, v, u\} \in \{[1 : D], [1 : V], [1 : U]\}$, respectively.

$$\mathbf{Y}[d, v, u] = \beta[d] + \sum_{c=1}^{C} \sum_{j=1}^{J} \sum_{k=1}^{K} \mathbf{X}[c, v + j - 1, u + k - 1] \times \Theta[d, c, j, k] \quad (1)$$

Fully-connected layers: The *fc* layer, which is a dense convolutional layer, is the classification layer. These layers can be seen as *conv* layers with no weight sharing (i.e W = K and H = J). Moreover, in a same way as conv layers, a non-linear function is applied to the outputs of *fc* layers. Each input neuron is connected to all output neurons (in Eq. 2).

$$\mathbf{Y}[u] = \beta[u] + \sum_{v=1}^{V} \mathbf{X}[v] \times \Theta[v, u] \quad (2)$$

Activation function: A nonlinear activation function is typically applied after each *conv* or *fc* layer. Various nonlinear functions are used to introduce non-linearity into the DNN, such as sigmoid, hyperbolic tangent (traditional activation

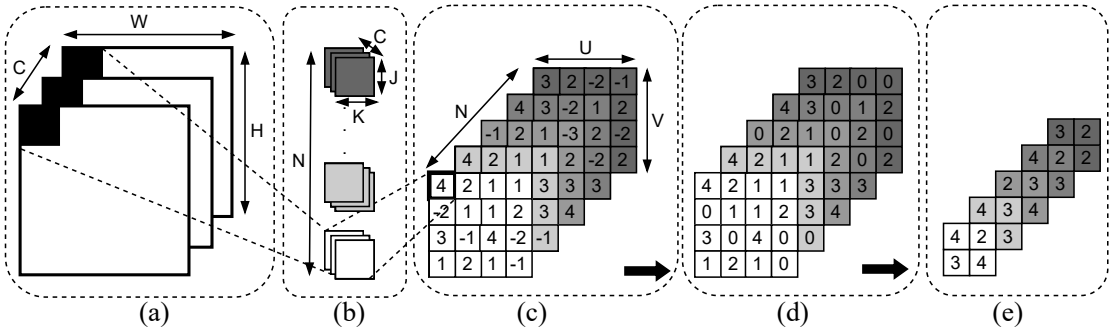

Figure 6. An example of the feed forward propagation in (convolution + ReLU) and max-pooling layers (Batch size B = 1, bias β omitted).
(a): Input feature maps (IFMs) in size of (C, H, W).
(b): Learned weight kernel in size of (D, C, J, K).
(c): Output feature maps (OFMs) in size of (D, V, U).
(d): OFM in size of (D, V, U) after activating by ReLU function.
(e): OFM in size of (D, V/2, U/2) after sub-sampling by max-pooling.

function), or the modern non-linear activation function is rectified linear unit (ReLU)], which has become popular in recent years due to its simplicity and its ability to enable fast training. Eq. 3 shows the rectified linear unit (ReLU) formula, which is exampled in Fig. 6(c) and (d).

$$\mathbf{Y}[d, v, u] = \max(0, \mathbf{X}[d, v, u]) \tag{3}$$

Max-pooling layers: A variety of computations that reduce the dimensionality of a feature map are referred to as pooling. Pooling, which is applied to each channel separately, enables the network to be robust and invariant to small shifts and distortions. Pooling combines, or pools, a set of values in its receptive field into a smaller number of values. It can be configured based on the size of its receptive field (e.g., 2×2) and pooling operation (e.g., max or average), as shown in Fig. 6(e) with max-pooling. As shown in Eq. 4, max-pooling of the input feature maps (IFMs) is the selection of the maximum element in a neighborhood $R \times R$ elements. As a result, the dimensionality of the feature maps is down-sampled. This layer helps to reduce the storage space and number of

computations for subsequent layers.

$$\mathbf{Y}[d, v, u] = \max_{p,q \in [1:R]} (\mathbf{X}[d, v + p - 1, u + q - 1]) \tag{4}$$

2.4 High Bandwidth Memory

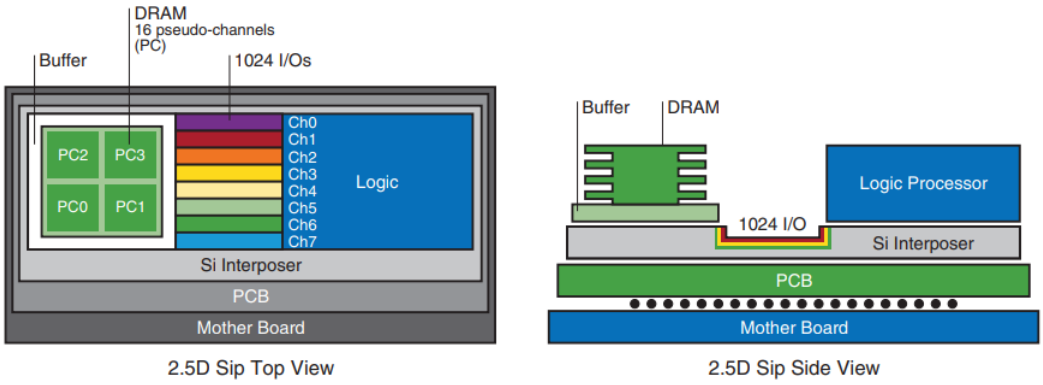

Figure 7. High Bandwidth Memory and 2.5D structure (Source: Samsung).

High bandwidth memory (HBM) uses a 3D-stacked die technology (through silicon via, TSV) to stack multiple interconnected DRAM chip. HBM is a high-speed and employs 2.5D system in package (SiP) memory technology [129, 130, 131, 132]. HBM2 (second-generation high bandwidth memory) allows to stack either four or eight DRAM dies on top of each other (See Fig. 7 [133, 134]). Multiple DRAM dies can stack on the same package. This significantly reduces the HBM package area compared to conventional multi-chip DRAM products. Furthermore, the closed DRAM stacks not only save the movement power of data, signals but also improve the movement speed due to a shorter distance.

Xilinx takes advantage of silicon stacking technologies of HBM to place the FPGA and the HBM DRAM beside each other in the same package. The result is a co-packaged HBM DRAM structure capable of multi-terabit per second bandwidth [130, 134]. This provides system designers with a significant step function improvement in bandwidth, compared to other memory technologies. Fig. 8 shows the communication between the HBM2 (physical) and the user logic (FPGA) [135]. Xilinx integrates two 4 GB HBM2 stacks (8 GB in total) and FPGA on the Alveo U280 card, divided into eight memory channels (MCs). Each MC is further divided into two 64-bit pseudo channels (PCs) occupying their own physical memory space. Each PC interacts with the FPGA via the standard advanced extensible interface (AXI). Two HBM2 stacks on the Alveo U280 card

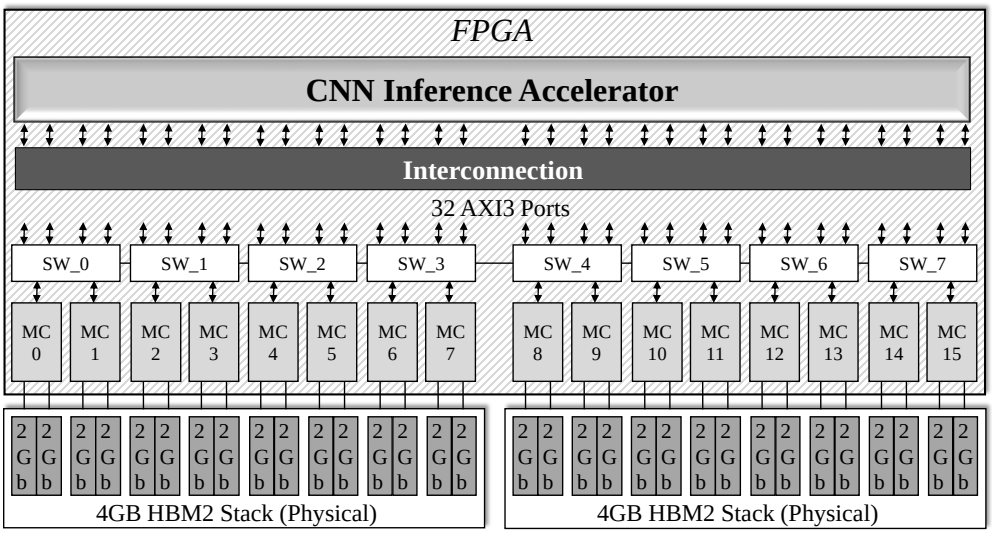

Figure 8. FPGA and HBM2 communication on Alveo U280 card.

consist of 32 AXI3 ports in total. An AXI3 port occupies a 2Gb, corresponding 256 MB (8 GB/32) memory region on HBM2. Note that, each AXI3 port can also access all memory regions on HBM2 via the crossbar switcher (SW) [136]. However, this takes more power and latency than using its own memory region. Furthermore, each HBM2 port operates at an independent clock rate.

2.5 Related Works

Many CNN inference accelerators have been developed. Their designs have focused on increasing the inference speed while reducing the power consumption to the lowest possible level. In this section, we present three drawbacks of the existing CNN inference accelerators.

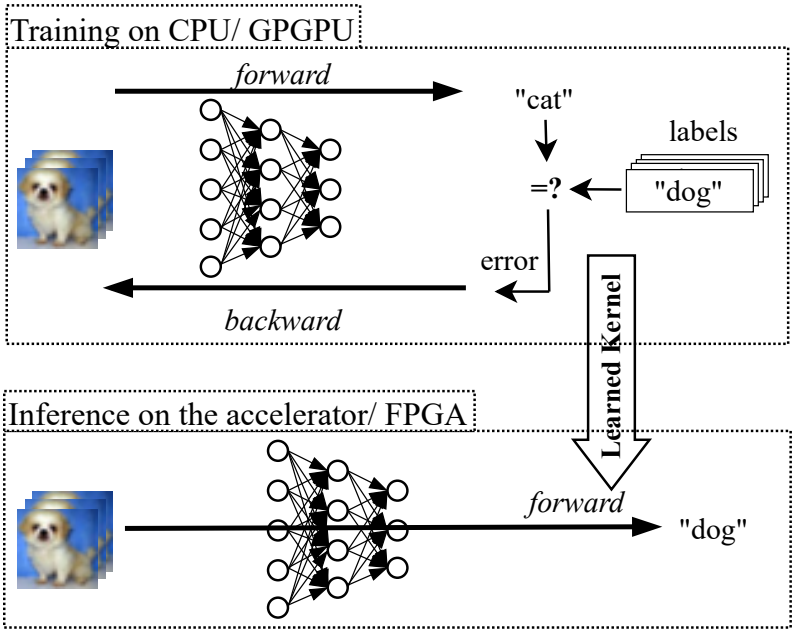

Figure 9. Current approach for the CNN accelerator.

First, the general-purpose graphic processing unit (GPGPU) is a well-known general-purpose hardware platform. Graphics processing units (GPUs) are originally developed to generate and display of 3D graphics. GPUs consist of massive processing cores and achieve high speed by paralleling the computing processors. Because of the high parallelism and processing capacity, the general-purpose utilization of GPUs is becoming a trend in many fields of high-performance computing [137]. The GPGPU is a good candidate for accelerating large-scale CNN-based applications by performing operations with massively parallel computational capacity. However, having all cores work in parallel leads to huge power consumption and poor power efficiency, which is the biggest challenge of the GPGPUs [55, 138]. Therefore, current approaches have been widely investigated FPGA-

based CNN inference accelerator due to its power efficiency benefits, besides taking advantage of GPGPU's high-performance computing for the training phase Fig. 9. In this research, we design an FPGA-based CNN inference accelerator in order to increase power efficiency. The accelerator has a multi-core architecture combined with shared-dual buffers to reduce external memory access due to broadcasting the same learned kernel among cores. In addition, the pipeline mechanism is used for inter-layers and inner of computation engines to speed up the inference phase.

Second, many studies have proposed FPGA-based techniques to improve the performance of CNN inference. For instance, calculation in the original CNN uses the number 32-bit floating-point, which gives a low calculation error but is very complicated. Therefore, most of reseachers developed FPGA-based CNN accelerators with low bit width by replacing the 32-bit floating-point with 32-bit fixed-point [66]. [63, 64, 68, 75, 84, 85, 86, 139] used 16-bit fixed-point, even [140, 141] utilized ternary and [67, 142, 143, 144, 145] replaced by binary computation unit. Low-bit-width computation significantly reduces computational and power costs, which of course has to be sacrificed in terms of accuracy. Other works [146, 147, 148] used on-chip memory for low external memory access and increasing reusability the learned kernel parameters. In addition, pipelined architecture [149, 150] has been explored to speed up inference time. Although these FPGA-based CNN inference accelerators significantly improve the performance of the accelerator, they still use double data rate memory (DDR) for the external memory to store the learned kernel parameters. Data transmission with the narrow bandwidth DDR between the external memory and the accelerator creates the bottleneck of the conventional CNN inference accelerator, especially with large-scale CNN models. To address this problem, we replace DDR memory with HBM2 as the external memory, which significantly ease the jam in learned kernel parameters transmission.

Third, R. Kuramochi et al. [151] used FPGA and HBM2 with high-level synthesis (HLS) to develop a CNN inference accelerator on the Alveo U50 platform. The latency of this proposal showed a 43% improvement compared to previous research. Their efficiency optimization [151] was obtained from both HBM2 and a randomly wired convolutional neural network (RWCNN) model. However, the

contributions of the HBM2 or RWCNN model was not presented clearly. Furthermore, although that work is 1.98 and 5.53× better than NVIDIA RTX 2080 Ti GPGPU and Intel i7-8700K CPU, respectively, only the batch size of 1 is used in the comparison. The evaluation results should be compared to those of GPGPU and CPU at various batch sizes. In this work, we describe in detail HBM2 implementation with Verilog HDL level (not high-level synthesis) and make a fair comparison with DDR memory-based CPU/GPGPU at several batch sizes. In addition, the effect of HBM2 and DDR memory on the accelerator in terms of throughput and power efficiency is analyzed. Consequently, we can reveal the superiority and trade-off factors when replacing DDR memory with HBM2.

3 System Architecture

CNN is a multiple layer model, the layers are dependable each other. An inter-layer pipeline architecture helps the inference process running in parallel and continously, which increases overall inference throughput. However, this requires a large number of hardware resource and high bandwidth pressure for multiple-layer weights loading. We take advantage of high-bandwith off-chip memory to load weights in parallel for inter-layer computation. In this section, we present the fully-pipelined manner (pipeline in inter-layer and computation engine) and describe the structure of the proposed accelerator. Next, the strategy of the accelerator is explained in detail.

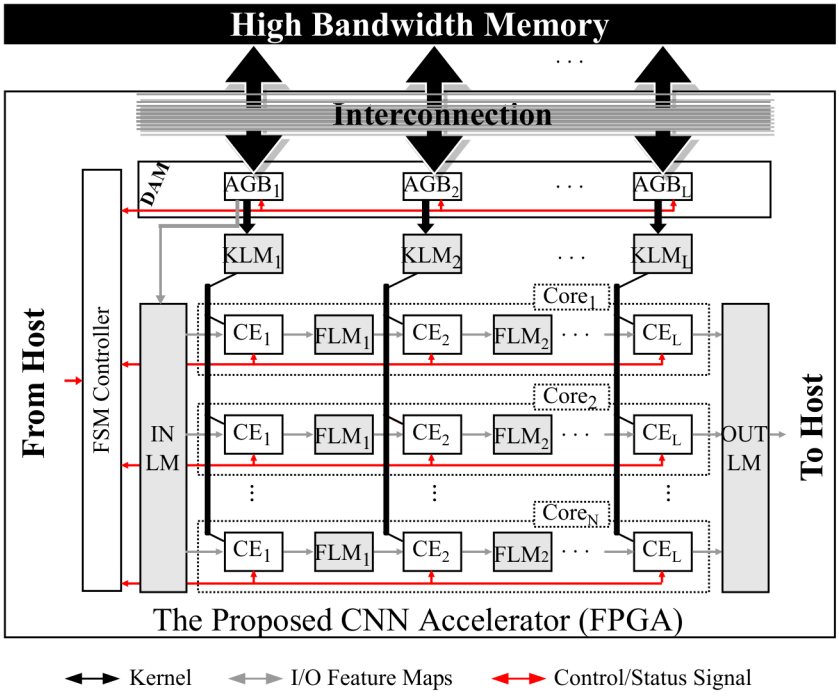

Figure 10. Overview of accelerator architecture.

3.1 Accelerator Architecture

The feed-forward propagation goes through L layers to get the final classified result. By using a pipeline mechanism between layers, the output generated in

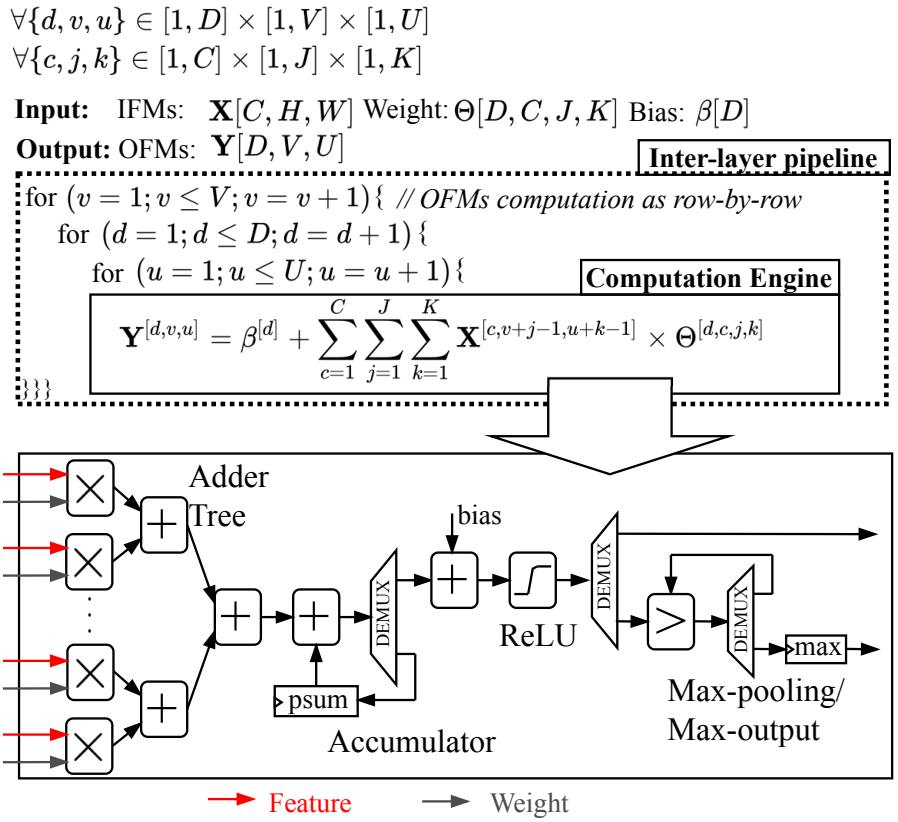

Figure 11. The pseudo code and hardware structure of convolutional computation engine.

the previous layer is immediately fed to the next layer (inter-layer pipeline). This means that all layers are executed in parallel, significantly reducing the latency for feed-forward propagation. The operation of each convolutional (*conv*) layer is detailed as pseudo code in Fig. 11. The output feature maps (OFMs) are calculated row by row, where v is incremented up to V. Note that the *fc* layer is dense of the *conv* layer. Therefore, OFMs at the *fc* layer are also computed as Fig. 11, which contains $\mathbf{X}[C,1,1], \Theta[D,C,1,1], \beta[D], \mathbf{Y}[D,1,1]$.

Fig. 10 shows an overview of the accelerator architecture, and Fig. 11 illustrates in detail the structure of the computation engine (CE). The proposed system uses the second-generation high bandwidth memory (HBM2) as the external memory to store all input images and learned kernel parameters. Accordingly,

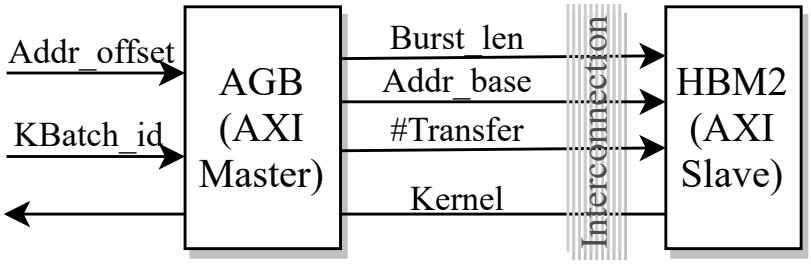

Figure 12. I/O signals of address generator block (AGB).

HBM2 communicates with the proposed CNN inference accelerator through multiple AXI buses. The accelerator uses a multi-core architecture (N cores, $Core_1$, ..., $Core_N$). The cores work in parallel and share the same dual kernel local memories (KLMs). Inside each core, there are L computation engines (CE_1, ..., CE_L). A CE has an associated dual feature local memory (FLM) to store that CE's OFMs and also feeds the IFMs to the next CE. The input images and kernel parameters are loaded into the dual local memory IMG LM and KLM in a batch-by-batch manner from HBM2. The data arbiter module (DAM) contains L address generator blocks (AGBs). They point to the memory region containing the kernel parameters in the HBM2, which the accelerator needs. Then, the DAM allocates input images and kernel parameters read from the HBM2 to the correct IMG LM and KLM, respectively.

The dual local memory (LM) is used inside the accelerator to store input images ($IN\,LM$), the classified results ($OUT\,LM$), the kernel parameters (KLM), and save input/output feature maps (FLM). This saves time instead of being stored in external memory. LMs are dual buffers (two banks), which operate in parallel in a ping-pong style. One bank is used for the writing process, while the other is for the reading process.

The data arbiter module (DAM) consists of L address generator blocks (AGBs) corresponding to L KLMs. Each AGB is assigned to a CE. The DAM communicates directly with the multiple-interface HBM2 by burst transmission via AXI4 interfaces. Each AGB also occupies multiple HBM2 ports. Depending on the kernel workload of each layer, the number of HBM2 ports is allocated differently at each AGB. The specific numbers are detailed in Section 4.2.

The learned kernel parameters of each layer are divided into $\#Kbatch$ batches and loaded to the corresponding KLM one by one. Then, the kernel batch is inputted to the respective CE. The capacity of the transferred kernel batch is $len(KBatch)$ bytes according to Eq. 5, where $len(Kernel\_layer)$ is the kernel parameter capacity of a layer. The $\#KBatch$ value is configured by the designer depending on the capacity of the KLM. The minimum capacity of the KLM must be $2 \times len(KBatch)$ bytes (dual-buffer). The kernel batches are loaded from the HBM2 into the KLM via $\#Port$ AXI4 ports in parallel. Then, each port loads *len(Workload)* bytes of data as in Eq. 6.

As input to AGB, $KBatch\_id$ (1, ..., $\#KBatch$) denotes which kernel batch the CE needs for the next computation and *Addr_offset* is the starting address each port accesses. This region is predefined by the finite state machine (FSM) controller. From there, AGB as the AXI master generates signals for AXI4 burst transmission according to Eq. 7, Eq. 8, Eq. 9, where $Burst\_len$ is 1, ..., or 256, $Addr\_base$ is reading address, $Data\_width$ is 256 bits and $\#Transfer$ is the number of burst transmissions by each port. Note that HBM2 is the AXI slave used to return kernel parameters to AGB Fig. 12.

$$len(KBatch) = \frac{len(Kernel\_layer)}{\#KBatch} \tag{5}$$

$$len(Workload) = \frac{len(KBatch)}{\#Port} \tag{6}$$

$$Burst\_len = 1, 2, 4, 8, 16, 32, 64, 128, 256 \tag{7}$$

$$Addr\_base = (KBatch\_id - 1) \times len(Workload) + Addr\_offset \tag{8}$$

$$\#Transfer = \frac{len(Workload)}{Data\_width \times Burst\_len} \tag{9}$$

The computation engine (CE) is the central computation block of the system, which is illustrated in Fig. 11. It executes the computations for the convolution, max-pooling, and fully connected layers, as shown in Eq. 1, Eq. 2, Eq. 3, Eq. 4 . CE consists of four phases working in pipelined style, which perform the multiplication, accumulation, activation function, and comparison operation. These phases are carried out as follows:

- Phase 1: Performing multiplications of multiple input features and kernels in parallel.
- Phase 2: Performing an additive calculation between the output results from Phase 1 according to the adder tree style. The cumulative total value is stored in the temporary register partial sum (*psum*). The convolutional result is the sum of the *psum* register and the bias value from the learned kernel.
- Phase 3: Activating the input by the ReLU function in Eq. 3. If the input is greater than or equal to 0, the output is the input itself. Otherwise, the output is 0.
- Phase 4: Determining the maximum value from a group of the input values. This phase is only enabled in the max-pooling and output layers (at the last *fc* layer to determine the output label). All CEs are synchronized and controlled by the FSM controller.

3.2 Workflow

$Core_1$, ..., $Core_N$ have completely the same structure and operate in parallel with separate input images but a shared kernel dual-buffer (KLM_1, ..., KLM_L). L computation engines work in a pipeline style in each core. Fig. 13 illustrates the timing diagram of the pipeline inference process of a core. The diagram shows execution of AGBs and CEs, which generate AXI4 signals to read the kernel parameters from HBM2 and to execute the computation, respectively.

The host PC starts the inference phase on the accelerator through the FSM controller. First, the input image is loaded and stored to IMG LM. Then, the CEs for the convolutional layer computes OFMs in a row-by-row ($R_{\#1}$, ..., $R_{\#V}$) manner as shown in Fig. 11. Because the number of kernel parameters for CONV1 is small, they are loaded once. From CONV2 onwards, kernel parameters are loaded in batches from HBM2 to the dual local memory during the computation for one row OFMs. After finishing the loading of a kernel batch, CE immediately starts the calculation using the kernel batch just loaded in KLM and the IFMs in FLM. While the CE processes the loaded kernel batch and the previously generated IFMs, AGB simultaneously loads the new kernel batch for the next computation. This operation is carried out until reaching the end of a row of OFMs, and then it continues row after row until completion of OFMs for that convolutional layer. Then CE begins to process a new input image. CEs for the fully connected layer work like a dense convolutional layer, and kernel parameters are also batch-loaded by batch and stored in the corresponding KLM. The CEs perform all four phases. The final output is saved to the *OUT LM*. At the end of the outputs for an input image batch, all results are sent back to the host PC by the DMA mechanism.

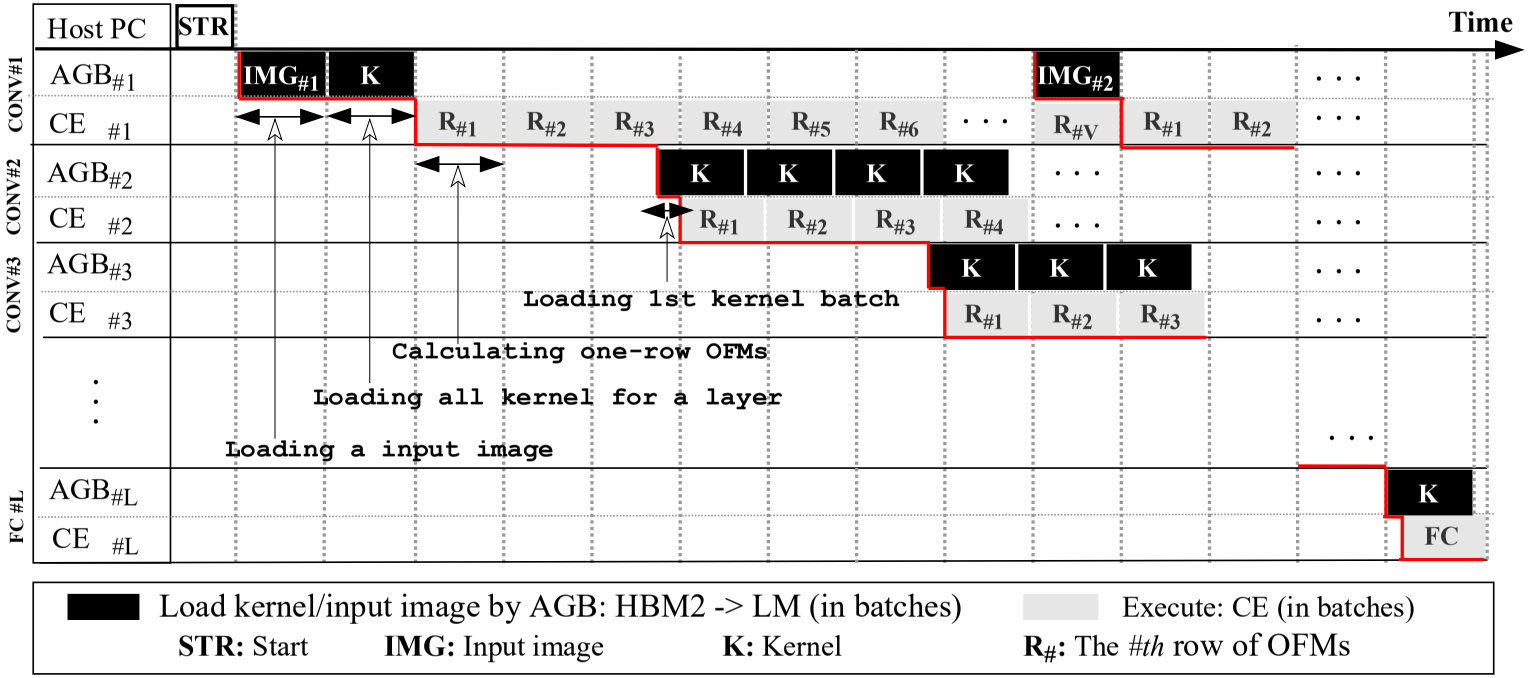

Figure 13. Timing diagram of inference phase of a core in the accelerator.

4 Experimental Setup

In this section, we first detail the parameters in this experiment, which are the size, workload, and computational cost at each layer of a typical CNN, VGG-16 model. Next, the system settings and the size of the accelerator are also described.

4.1 VGG-16 Model

Table 3. Features and performance of VGG-16 model in our experiment.

Model	**Dataset**	**#Parameter**	**#Operation**	**Top-1 acc**	**Top-5 acc**
VGG-16	CIFAR100	15.29M	0.63G	91.2%	98.6%
	ImageNet	138.36M	30.96G	73.4%	91.5%

M: Million ($\times 10^6$), G: Giga ($\times 10^9$), (batch size: B=1).

A typical CNN, the VGG-16 model is explored with different sizes, which is described in Table 2. The VGG-16 model includes typical layers with 13 convolutional (*conv*) layers, three fully connected (*fc*) layers, and five max-pooling (*pool*) layers. After each convolutional layer and fully connected layer, the activation function is executed. The order of layers in the VGG-16 model is detailed in Table 3. The max-pooling layer is placed right after the *conv2*, *conv4*, *conv7*, *conv10*, and *conv13* layers. The number of learned parameters in each layer is the sum of the corresponding learned weights and learned biases in that layer. Table 3 breaks down the number of kernel parameters for convolutional layers (*#p_conv*) and fully connected layers (*#p_fc*), as specified in Eq. (10)-(11). There is no learned parameter in the *pool* layer. Totally, the number of learned parameters is approximately 15.3 and 138.4 million with small and large-scale models, respectively.

$$\#p\_conv = (D \times C \times J \times K) + D \tag{10}$$

$$\#p\_fc = (V \times U) + D \tag{11}$$

Table 4 also shows the number of fixed-point 16-bit operations at each layer. As mentioned in a previous work [152], each feature in OFMs requires $K \times K$ multiply-and-accumulate (MAC) operations (J=K). Here, a MAC includes the

Table 4. Breakdown VGG-16 model.

	Function	**Small Scale CNN Model (w/ CIFAR100)**		**Large Scale CNN Model (w/ ImageNet)**	
		Parameter	Operation	Parameter	Operation
		No. ($\times 10^6$)/Percent	No. ($\times 10^6$)/Percent	No. ($\times 10^6$)/Percent	No. ($\times 10^6$)/Percent
CE1	conv1/ReLU	0.002/ 0.01	3.6/ 0.57	0.001/ 0	176.6/ 0.57
CE2	conv2/ReLU/pool	0.037/ 0.24	75.6/ 12	0.037/ 0.03	3705/ 12.01
CE3	conv3/ReLU	0.074/ 0.48	37.78/ 6.02	0.074/ 0.04	1851/ 5.98
CE4	conv4/ReLU/pool	0.148/ 0.97	75.56/ 12	0.148/ 0.11	3702/ 12
CE5	conv5/ReLU	0.295/ 1.93	37.77/ 6.01	0.295/ 0.21	1850/ 5.98
CE6	conv6/ReLU	0.59/ 3.86	75.5/ 12	0.59/ 0.43	3700/ 12
CE7	conv7/ReLU/pool	0.59/ 3.86	75.53/ 12	0.59/ 0.43	3701/ 12
CE8	conv8/ReLU	1.18/ 7.72	37.76/ 6.01	1.18/ 0.85	1850/ 5.98
CE9	conv9/ReLU	2.36/ 15.4	75.5/ 12	2.36/ 1.71	3700/ 12
CE10	conv10/ReLU/pool	2.36/ 15.4	75.51/ 12	2.36/ 1.71	3700/ 12
CE11	conv11/ReLU	2.36/ 15.4	18.88/ 3.01	2.36/ 1.71	925/ 2.99
CE12	conv12/ReLU	2.36/ 15.4	18.88/ 3.01	2.36/ 1.71	925/ 2.99
CE13	conv13/ReLU/pool	2.36/ 15.4	18.88/ 3.01	2.36/ 1.71	925/ 2.99
CE14	fc1/ReLU	0.263/ 1.72	0.524/ 0.08	102.76/ 74.3	206/ 0.66
CE15	fc2/ReLU	0.263/ 1.72	0.524/ 0.08	16.78/ 12.1	33.5/ 0.11
CE16	fc3/ReLU/output	0.051/ 0.34	0.102/ 0.02	4.1/ 2.96	8.2/ 0.03
All		**15.29/ 100**	**628/ 100**	**138.36/ 100**	**30959/ 100**

two operations of multiplication and addition. The ReLU function consumes only one comparison operation. Accordingly, the number of operations in a convolutional layer ($\#op\_conv$) is given by Eq. (12). Mathematically, the number of operations in the fully connected layer (($\#op\_fc$)) can be computed as Eq. (13). The max-pooling layers in CNNs summarize the outputs of neighboring neurons to reduce feature dimensions. As shown in Eq. (14), the number of comparison operations is $\#op\_pool$. We select the maximum among neighboring $R \times R$ *(R=2)* elements, which three comparison operations are used for one output at the *pool* layer. In total, there are about 628 million and 30.96 billion fixed-point 16-bit operations, including additions, multiplications, and comparisons, with small and large-scale models, respectively. Note that Table 3 deals with a batch size of 1.

$$\#op\_conv = (D \times C \times V^2 \times K^2 \times 2) + (D \times V^2) \tag{12}$$

$$\#op\_fc = U \times W \times 2 \tag{13}$$

$$\#op\_pool = D \times V \times U \times 3 \tag{14}$$

4.2 Experimental Setup

In this section, the environment setup and the strategy for system verification are presented in detail.

The CUDA Basic Linear Algebra Subroutine (CuBLAS) library is used on NVIDIA GeForce RTX 3090 GPGPU (2666 MHz core frequency) and 256 GB DDR4 RAM for the training phase. We use the CIFAR-100 dataset [153] for the experiment, which includes a total of 60,000 tiny color images with a size of $3 \times 32 \times 32$. Of these, 50,000 images are used for training, and the remaining 10,000 images are used for testing. This dataset has 100 classes.

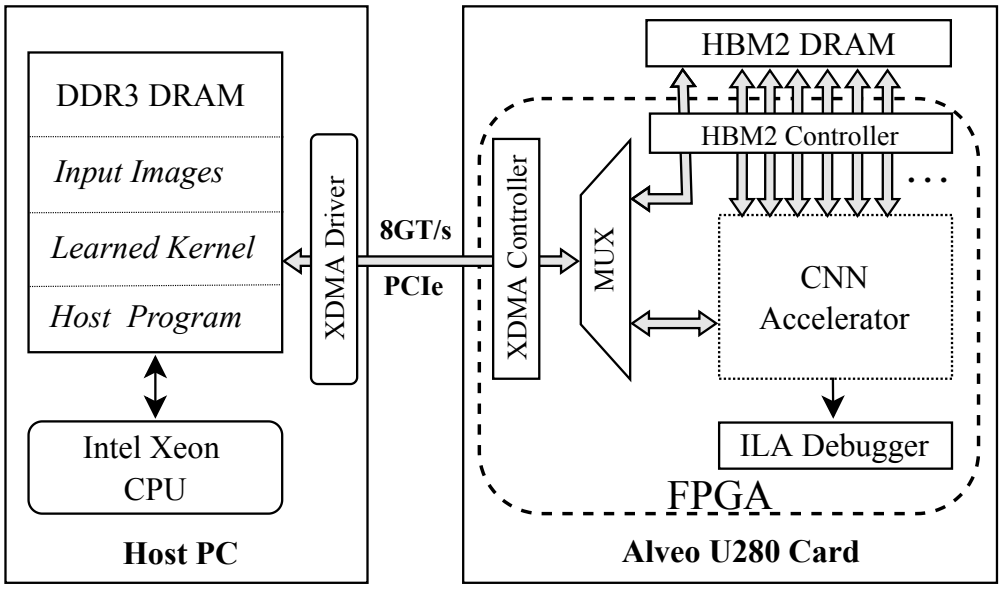

Figure 14. Experimental system setup.

Fig. 14 shows an overview of the experimental setup for system verification. The proposed CNN accelerator is implemented and verified on a Xilinx FPGA Alveo U280 Data Center Accelerator Card (16 nm), which includes 8 GB of second-generation high bandwidth memory (HBM2) DRAM and FPGA components. The FPGA on the Alveo U280 card includes more than 1,300k look-up tables (LUTs), 2,600k flip-flops (FFs), 2,000 36kb block RAMs (BRAMs), 960 ultra RAMs (URAMs), and 9,024 digital signal processors (DSPs). The Alveo U280 card communicates to the host PC by the Peripheral Component Interconnect

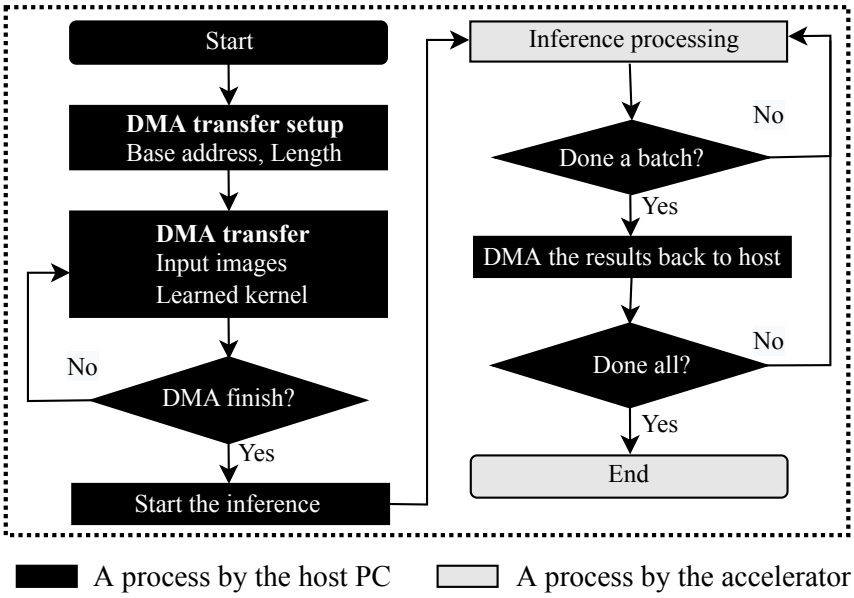

Figure 15. High-level flowchart of the system's strategy.

Express (PCIe) 3.0 protocol with 8 GT/s (giga transfer per second) at maximum performance. The host PC integrates an Intel Xeon CPU E5-2620v2@2.10 GHz and 128 GB DDR3 memory, which stores input images, learned kernel parameters, and the host program. The host CPU runs on CentOS 7.9. Due to the FPGA resource constraint on the Alveo U280 card, the accelerator architecture consists of four cores ($N = 4$), each core consisting of 16 computation engines ($L = 16$). Table 3 shows that the majority of calculations are performed at CE2 through CE13. Therefore, we design CEs with different computing abilities. Specifically, CE1 contains $3 \times 3 \times 3 = 27$ multipliers, 28 adders, two registers, and two comparators. CE2, CE3, ..., CE13 each includes $8 \times 3 \times 3 = 72$ multipliers, 73 adders, two registers, and two comparators. Each of the three remaining CEs (for three *fc* layers) contains 64 multipliers, 65 adders, two registers, and two comparators.

The CUDA Basic Linear Algebra Subroutine (CuBLAS) library is used on NVIDIA GeForce RTX 3090 GPGPU (2666 MHz core frequency) and 256 GB DDR4 RAM for the training phase. We explore both the CIFAR-100 dataset [153] with tiny color images ($3 \times 32 \times 32$, 100 classes) and ImageNet dataset [**?**]

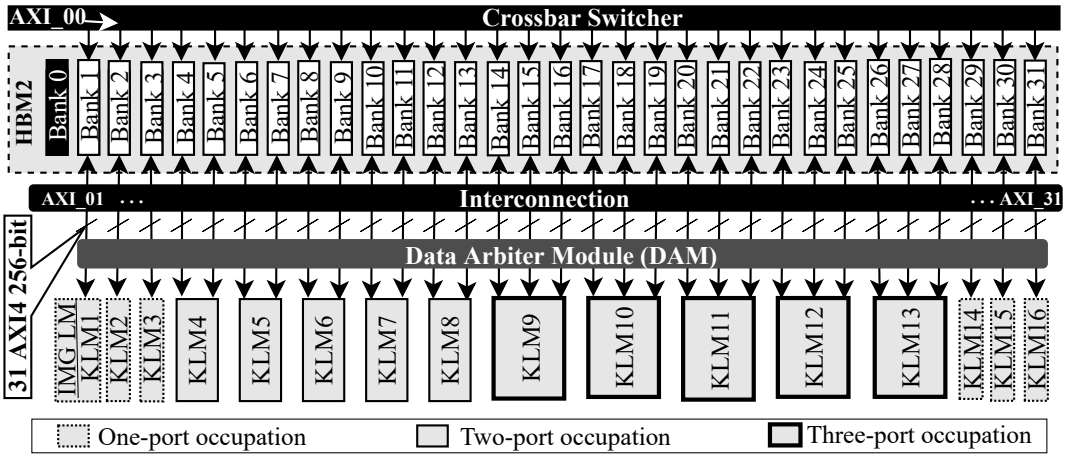

Figure 16. Data allocation on the HBM2.

with large scale images ($3 \times 224 \times 224$, 1000 classes) in this experiment. The datasets include a total of 60,000 with 50,000 images used for training, and the remaining 10,000 images are used for testing.

The flowchart in Fig. 15 describes how the system works. The learned kernel parameters and input images are initially stored on the DDR3 memory in the host PC. Before the accelerator starts the inference processing, all of these learned kernel parameters and input images are transmitted down to the Alveo U280 card and stored in the HBM2 by the Xilinx Direct Memory Access (XDMA) module. After all input images and learned kernel parameters are saved in the HBM2 on the Alveo U280 card, the host PC activates the inference process on the proposed accelerator via the FSM controller. When the accelerator is enabled, it operates as described in Section 3.2. The classification results are output in batches and sent back to the host PC by DMA manner. Then they are compared with correct labels in software program (written in C language). This process is repeated until the accelerator classifies all the testing images.

Fig. 16 illustrates the data allocation in the HBM2 in this experiment. The 8 GB HBM2 on the Alveo U280 card is divided into 32 partitions. Each partition has a capacity of 256 MB (8 GB/32), which is occupied by one AXI port. That means 32 partitions are occupied by 32 AXI ports. The XDMA controller takes up one port of the HBM2 (AXI_00 interface) to perform the DMA process, which allocates the input images and learned kernel from the host PC to the 31 remain-

ing HBM2 partitions via an AXI_00 interface. The remaining 31 ports (AXI_01, AXI_02, ..., AXI_31) of the HBM2 are occupied by the accelerator. These ports are allocated to load the input images (*IN LM*) and the learned parameters for the 16 layers of the VGG-16 model. The accelerator consists of 16 CEs, corresponding to 16 KLMs. Based on the resource constraint and the kernel capacity of each layer (calculated as #Parameter $\times$ 16-bit), we configure the $\#KBatch$ value as in Table 4 in this experiment, where the minimum capacity of the KLM in each layer must be $2 \times len(KBatch)$ bytes (dual-buffer) (see Section 3.1). Particularly at *conv1*, *#KBatch* is 1 so KLM capacity is only *len(KBatch)* bytes.

As for allocating the number of HBM2 ports to a KLM, we split them into three groups, where each KLM has one, two, or three HBM2 ports, as illustrated in Fig. 16 and described in Table 5, where *#KBatch/one row of OFMs* and *#KBatch/ image* columns denote the number of DAM's requests to load kernel parameters at the respective layer to calculate for one row of OFMs and for all OFMs (consist of V rows), respectively. We break down the small/large scale VGG-16 model as layer by layer in Table 4 (*Parameter* column). At *conv* layers, the number of HBM2 ports is allocated to each CE based on the number of parameters that need to be loaded from HBM2. Here, AGB1, AGB2, AGB3 are allocated one HBM2 port/AGB; from AGB4 to AGB8 are allocated two HBM2 ports/AGB; from AGB9 to AGB13 are allocated three HBM2 ports/AGB. At *fc* layers, we have reduced the pressure of accessing off-chip memory by reusing loaded parameters to all inputs. The loaded kernel batch is held for computing all input values before loading a new kernel batch. Besides, the temporary outputs are stored in FLM for the next computation with the next kernel batch. Hence, even though *fc* layers take up most of the parameters ($\sim$90% in large scale CNN model), we only use one HBM2 port/AGB, instead of allocating a lot of HBM2 ports to AGB at *fc* layers. Each CE occupies one KLM. Each KLM consists of one or multiple HBM2 ports (see Fig. 15 and Table 5). Each port works independently to avoid collisions with each other and among CEs. The width of the data bus is 256 bits. Applying the optimized strategy allocation in HBM2 in an earlier work [**?**] optimized the latency and power to transmit data from the HBM2 to the accelerator through the individual ports. Ports can also access the data partition of the other ports, but it will take more latency and power because

they have to go through the crossbar switcher to arbitrate. Therefore, the learned parameters of each layer will be allocated by the host program to the partition closest to the ports occupied by that layer.

An interconnection module is used to connect the HBM2 (AXI3 interface) with the accelerator (AXI4 interface). The communication between the HBM2 (physical) and the FPGA elements is managed by the HBM2 controller. Furthermore, to clarify the impact of HBM2 and DDR memory on the accelerator in terms of performance and power efficiency, we replaced HBM2 with DDR4 memory on the Alveo U280 card with the same accelerator architecture. Accordingly, the FPGA resources consumed on the accelerator are less when using DDR4 compared to HBM2 (31 vs. 1 AXI interface, 16 vs. 1 AGB).

5 Experimental Results

In this section, the throughput and power consumption results of the proposed accelerator are presented. The dynamic power consumption is obtained using the Xilinx Power Estimator tool. The RTL design is built with Verilog HDL. We synthesize the bitstream for the hardware design using Vivado 2019.2, which also presents hardware resources after the implementation step. The throughput is measured by giga operations per second (GOP/s), which is calculated by Eq.(15), where $\#Operation$ is the number of calculations performed, T_{FLD} (time for the first load) denotes the latency for loading the first input image batch and the learned kernel parameter for *conv1* layer from HBM2 to local memory), T_{INF} indicates the inference time of the accelerator, and T_{DRAIN} is latency for backing (by DMA) classification result from buffer to the host PC. Notice that the time is measured in seconds (s) and does not include the latency needed to transmit the input images and kernel parameters from the host PC to HBM2 by the DMA mechanism.

$$Throughput = \frac{\#Operation(GOP)}{(T_{FLD} + T_{INF} + T_{DRAIN})(s)} \tag{15}$$

Meanwhile, the power efficiency is measured by giga operations per second per watt (GOP/s/W). As shown in Eq. (16), the power efficiency is calculated as the throughput divided by the dynamic power consumed by the system.

$$Power\_efficiency = \frac{Throughput(GOP/s)}{Power(W)} \tag{16}$$

We maintain the accelerator architecture and use HBM2 and DDR4 DRAM memory on the Alveo U280 card for the off-chip memory, respectively. With this implementation, we can see the difference in the effect of HBM2 and DDR4 memory on the accelerator in terms of memory bandwidth. The memory bandwidth (BW) is measured as shown in Eq. (17), which is the maximum amount of data exchanged between the off-chip memory and the accelerator in one second (GB/s). Here, $len(A\_burst\_transfer)$ (bytes) is the number of data transferred in a burst transmission of each port, which is the product of the data bus width ($Width\_data$) and the number of sequential data transferred ($Burst\_len$). BW is the product of the amount of data transferred in one clock cycle $\frac{Len(A\_burst\_transfer)}{\#Clock\_cycle}$,

the maximum number of interfaces that can be executed simultaneously ($\#Port$, 31 for HBM2 and 1 for DDR4 memory), and the operating frequency of the system (Fq).

$$\begin{aligned} BW &= \frac{Len(A\_burst\_transfer)}{\#Clock\_cycle} \times \#Port \times Fq \\ &= \frac{Width\_data \times Burst\_len}{\#Clock\_cycle} \times \#Port \times Fq \end{aligned} \tag{17}$$

5.1 Results of the CNN Inference Accelerator

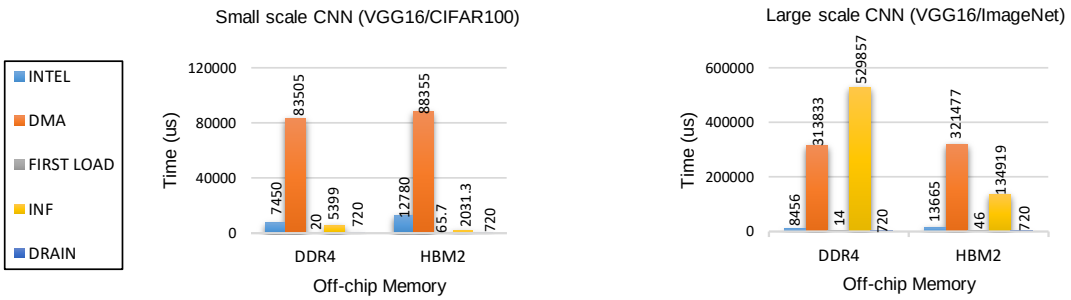

Figure 17. Breakdown timing for the inference process.

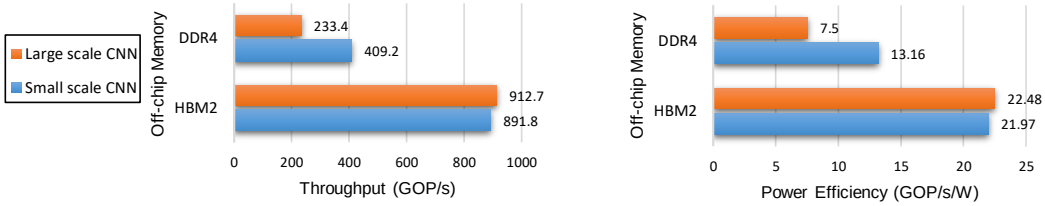

Figure 18. The proposed CNN inference accelerator results in small and large scale CNN model:
Throughput (GOP/s) (left) and power efficiency (GOP/s/W) (right).

The proposed architecture contains $\#Mult$ multipliers between input feature x and weight w at each core. Each DSP takes one clock cycle for each calculation at 250MHz frequency. In the ideal state, assuming the weights w and input feature x

Table 5. Configuration for VGG-16 hardware implementation
(#Mult for one core, #Port for shared dual local memories (KLMs)).

	#Mult	#Port	Small Scale CNN Model		Large Scale CNN Model		Ideal_BW (GB/s)	Max_BW (GB/s)	KLM size (kB)
			#KBatch/ row of OFMs	#KBatch/ image	#KBatch/ row of OFMs	#KBatch/ image			
CE1	27	1	1	32	1	224	13.5	3.83	3.5
CE2	72	1	4	128	4	896	36	3.83	36.063
CE3	72	1	8	128	8	896	36	3.83	36.063
CE4	72	2	16	256	16	1792	36	6.94	36.032
CE5	72	2	16	128	16	896	36	6.94	72.063
CE6	72	2	32	256	32	1792	36	6.94	72.032
CE7	72	2	32	256	32	1792	36	6.94	72.032
CE8	72	2	32	128	32	896	36	6.94	144.063
CE9	72	3	64	256	64	1792	36	9.52	144.032
CE10	72	3	64	256	64	1792	36	9.52	144.032
CE11	72	3	128	256	128	1792	36	9.52	72.016
CE12	72	3	128	256	128	1792	36	9.52	72.016
CE13	72	3	128	256	128	1792	36	9.52	72.016
CE14	64	1	16	16	1566	1566	32	3.83	64.125[a]/256.5[b]
CE15	64	1	16	16	1023	1023	32	3.83	64.125
CE16	64	1	4	4	320	320	32	3.83	50.098
All	**1083**	**31**					**541.5**	**105.28**	**1154.3[a]/1346.7[b]**

[a] For small-scale CNN model (w/ CIFAR100 dataset)

[b] For large-scale CNN model (w/ ImageNet dataset)

Table 6. Experimental results on the proposed accelerator with DDR4/HBM2 off-chip memory.

Off-chip Memory		DDR4	HBM2
CNN Architecture		4×16 CEs, 1 AGB	4×16 CEs, 16 AGBs
FPGA Platform		Xilinx Alveo U280	
Frequency (MHz)		250	
Precision		16-bit fixed-point	
#FFs		1,357K (52%)	1,697K (65%)
#LUTs		412K (32%)	721K (55%)
#DSPs		8,684 (96%)	8,681 (96%)
#BRAMs		1,803 (89%)	1,861 (92%)
#URAMs		874 (91%)	897 (93%)
Power (W)		31.1	40.6
Latency (ms/image)	Tiny image (3×32×32)	1.535	0.704
	Large image (3×224×224)	132.648	33.921

are always available for computation every clock cycle, then the ideal bandwidth for the proposed accelerator would be calculated by the following formula:

$$Ideal\_BW = N\_bit \times \#Mult \times Freq \tag{18}$$

where *N_bit* (16-bit fixed-point) and *Freq* denote the length of each value and the operating frequency, respectively. Note that cores work in parallel and share the same weight, so here we only consider *#Mult* of a core.

The ideal bandwidth of each CE is shown in Table 4. We have a total of the required bandwidth in the ideal state at 250MHz frequency which will be 541.5GB/s (16×1083×250MHz). However, with HBM2 off-chip memory, the theoretical maximum bandwidth can reach 460GB/s at 900Mhz frequency, or 127.78GB/s equivalent to ∼4GB/s each port at 250GB/s frequency when all 32

ports are used up. In fact, the total maximum bandwidth is only 105.28GB/s, corresponding to 82.4% on average compared to the theoretical maximum bandwidth (at 250MHz frequency). The first cause of bandwidth degradation is that the proposed accelerator only uses 31 ports instead of the full 32 ports. Second, CEs that use two or three HBM2 ports, the actual bandwidth is achieved by 86.8% (6.94 vs. 4GB/s $\times$ 2) or 79.3% (9.52 vs. 4GB/s $\times$ 3) compared to the ideal, respectively. That means the experimental bandwidth does not increase linearly when increasing the number of ports/CE. Formula (5)-(9) describe how to calculate $\#Transfer$, which is the number of burst transmissions by each port. There is always a delay between burst transmissions (config and transmit). To maximize transmission performance, *Burst_len* should be held at 256 as Eq. (7). In the proposed accelerator, however, len(KBatch) is divided by each port, *len(Workload)* (as Eq. (6)) is not always kept *Burst_len* at the maximum value (256). This is the second overhead resulting in bandwidth degradation. Overall, the experimental bandwidth achieved in this system is 19.44% (105.28 vs. 541.5GB/s) of the required bandwidth when using HBM2 off-chip memory, and only 1.5% (7.95 vs. 541.5GB/s) if using DDR4 memory (as Table 5).

We break down the execution time of each stage in Fig. 8. Which includes INTEL indicating the CPU host processing time; DMA represents the latency for allocating all kernel parameters and input images (all small scale images or a batch large scale images) from DDR3 (host PC) to HBM2 (Alveo U280) via PCIe+AXI_00 by DMA mechanism; FIRST LOAD is the latency to load a batch of input small scale image (or first three-row for large scale images) and kernel parameters for *conv1*; INF denotes latency for inference phase (accelerator) and DRAIN represent a time for backing results to host PC.

The proposed accelerator takes advantage of the large hardware resources from the Alveo U280 FPGA and the feature of sharing kernel parameters among cores to increase the computational ability of the accelerator. From Table 5, the proposed system takes up 65% FFs, 55% LUTs, 92% BRAMs, 93% URAMs and 96% DSPs of the FPGA resource on the Alveo U280 card. Meanwhile, the proposed accelerator with DDR4 memory takes less 13% FFs and LUTs, 2-3% URAMs and BRAMs than with HBM2. The reason for this difference is the use of more AXI4 interfaces in the system to communicate with the off-chip memory (31

vs. 1 interface), which means the interconnection takes more hardware resources and DAM uses more AGB (16 vs 1).

Fig.9 illustrates the performance of the proposed system. The peak throughput can be approximated by the formula below:

$$Peak\_Throughput = 2 \times \#Mult \times Freq \tag{19}$$

where index *2* is added for calculations for matrix multiplication (multiplication and accumulation), *#Mult* is applied to 4 cores, *Freq* denotes the operation frequency. In the ideal case (executing continuously at every clock cycle with the ideal memory bandwidth being 541.5 at 250 MHz frequency) the ideal throughput can be estimated at 2.166 TOP/s as Eq.(19). The actual throughput achieves 912.7 GOP/s or 891.8 GOP/s for the large/small scale model, which corresponds to 42.1% or 41.2% of the ideal case, respectively. Although the experimental bandwidth drops ∼5× (105.28 vs. 541.5 GB/s) compared to the ideal, which significantly degraded the system throughput. However, taking advantage of the ability to share kernel parameters between cores (×4 core) has improved the overall throughput of the accelerator (only drops ∼2.4× from the ideal throughput). Thanks to the higher reusability of loaded parameters from HBM2 for computation of the large scale model compared to the small scale model, we can observe a slight difference in throughput (912.7 vs. 891.8 GOP/s) when implementing the two models with different sizes from Fig.9.

5.2 Analysis of Limitation and Discussion

In this study, we propose a multi-core accelerator combined with a dual shared local memory containing kernel parameters to maximize the computing ability and reduce latency to read kernel parameters from HBM2 off-chip memory. Despite the positive results in terms of throughput and power efficiency, we observe the following weaknesses in the current system, which will be improved in the future.

First, the accelerator only uses an average of 82.4% (105.28 vs. 127.78 GB/s) of HBM2's maximum bandwidth at 250 MHz frequency. The solution for improvement would be to utilize all ports (32 ports instead of just 31 ports). In

addition, burst transmission performance at each HBM2 port is maximized by always keeping *Burst_len* at 256.

Second, our proposed accelerator is currently taking full advantage of the hardware resource available on the Alveo U280 FPGA (96% DSP, 92% BRAM, 93% URAM) to increase the computation and parallelism ability of the inference process (multi-core, shared local memory and fully pipeline). This results in high power consumption up to 40.6W. While the maximum experimental bandwidth of HBM2 off-chip memory can only achieve $\sim$1/5 the required bandwidth (ideal state) from the accelerator. A balance between computing ability and response bandwidth to load kernel parameters from off-chip memory will guide the next improvement.

Third, the disadvantage of the fully pipeline structure is that accelerators occupy a large amount of computational and storage resources because CEs are computed in parallel and intermediate results are cached among different layers. In this study, we have implemented VGG16 model for the experiment due to hardware resource constraint. With CNN models with more than 16 layers (i.e., VGG19, ResNet50), are being researched to use layer-folding pipeline structure [154] in the future.

5.3 Comparison with Other DDR-based Works

Table 7 shows the correlation power consumption, inference performance, and power efficiency between the proposed system and the existing FPGA-based accelerators. We selected existing FPGA and DDR memory-based accelerators, deployed on the same architecture model (VGG-16) and the large-scale ImageNet dataset, for a fair comparison. They have been measured to the same units for a fair comparison of throughput (GOP/s) and power efficiency (GOP/s/W).

The throughput of our experimental accelerator using DDR4 memory is only 1.7× better and even 1.3× worst than the throughput of two earlier accelerators [84], [85], respectively. Moreover, these accelerators surpass in terms of power efficiency. However, if the proposed accelerator uses HBM2 off-chip memory, the throughput is 6.7× and 3× better, respectively. Furthermore, in terms of power efficiency, the proposed system is 1.6× and 1.7× better. Ma et al. [86] proposed an accelerator with throughput that is 1.8× (1,605 vs. 912.7 GOP/s) faster than our proposed system, but there is a tradeoff in power efficiency, making their system 1.4× (16.1 vs. 22.48 GOP/s/W) worse than ours in this important factor.

5.4 Comparison with Other HBM2-based Works

Kuramochi et al. [151] proposed an FPGA-based CNN inference accelerator, which is implemented on the Xilinx Alveo U50 card. Although that system was explored as parallel architecture (for a randomly wired convolutional neural network, RWCNN) and used HBM2 for the off-chip memory, our proposed system is 1.9× (912.7 vs. 474.4GOP/s) better in throughput at a frequency of 250 MHz. Even if we assume a decrease in the frequency to 200 MHz (equal to their value [151]), our proposed system's throughput is still 1.5× better.

Table 7. Comparison with previous FPGA-based CNN inference accelerators.

	IEEE Trans. (2018) [84]	FCCM (2019) [85]	IEEE Trans. (2020) [86]	FPL (2020) [151]	**Our proposal**
Model	VGG-16	VGG-16	VGG-16	RWCNN	**VGG-16**
Dataset	ImageNet	ImageNet	ImageNet	ImageNet	**ImageNet**
FPGA Platform	Zynq XC7Z045	Zynq ZCU102	Stratix 10 GX 2800	Alveo U50	**Alveo U280**
Off-chip Memory	DDR DRAM	DDR DRAM	DDR DRAM	HBM2 DRAM	**HBM2 DRAM**
Precision	16-bit fixed-point	16-bit fixed-point	8/16-bit fixed-point	16-bit fixed-point	**16-bit fixed-point**
Frequency (MHz)	150	200	300	200	**250**
Power (W)	9.63	23.6	100	-	**40.6**
Throughput (GOP/s)	137	309	1,605	474.7	**912.7**
Power Efficiency (GOP/s/W)	14.2	13.1	16.1	-	**22.48**

5.5 Comparison with DDR-based CPU/GPGPU

Since the existing FPGA-based CNN inference accelerators have poor performance and low flexibility, the proposed system needs to be evaluated with other high-performance and high-flexibility hardware platforms such as CPU/GPGPU. The accelerator of Kuramochi et al. [151] was also compared with CPU and GPGPU in throughput, but the comparison was only with a batch size of 1. GPGPU can achieve high efficiency with a larger batch size because of its powerful parallel computing ability. Therefore, in this section, we make a fair comparison of the proposed system with Intel i9-10940X CPU and NVIDIA GeForce RTX 3090 GPGPU in terms of throughput and power efficiency at various batch sizes. The system configurations of CPU and GPGPU are given in Table 8, and the comparison is illustrated in Fig. 19 for throughput and power efficiency among the DDR-based CPU/GPGPU and the proposed system with both HBM2 and DDR-based memory. Since the proposed accelerator has 4 cores, a batch size of 4 is the maximum.

Table 8. Environment setup for CPU and GPGPU platform.

Device	Intel i9-10940X CPU	NVIDIA GeForce RTX 3090 GPGPU
Clock rate used	3.2-4.1GHz	1.74GHz
Memory	8×32GB DDR4 2666MHz	
OS	CentOS 7.9	Driver 470.57.02
Compiler	gcc 4.8.5	CUDA 11.2
Python	Pytorch/python3	
BLAS Library	OpenBLAS	cuBLAS

The throughput of the proposed system (with HBM2) and that of the GPGPU are roughly equivalent at batch sizes of 1 and 2 (Fig. 19(a) and 19(c)). Compared to the CPU, ours is 6× (223 vs. 37) and 2.8× (446 vs. 160) better in throughput with the small-scale CNN model, respectively. With the large-scale CNN model, the proposed accelerator even achieves 8.4× (228 vs. 27 GOP/s) and 9.5× (456 vs. 48 GOP/s) at batch sizes of 1 and 2, respectively. At a batch size of 4, our system gains a peak throughput, which is 1.3×/1.65× and 3.3×/11.9× better than GPGPU and CPU with the small/large-scale models, respectively. The graphs in

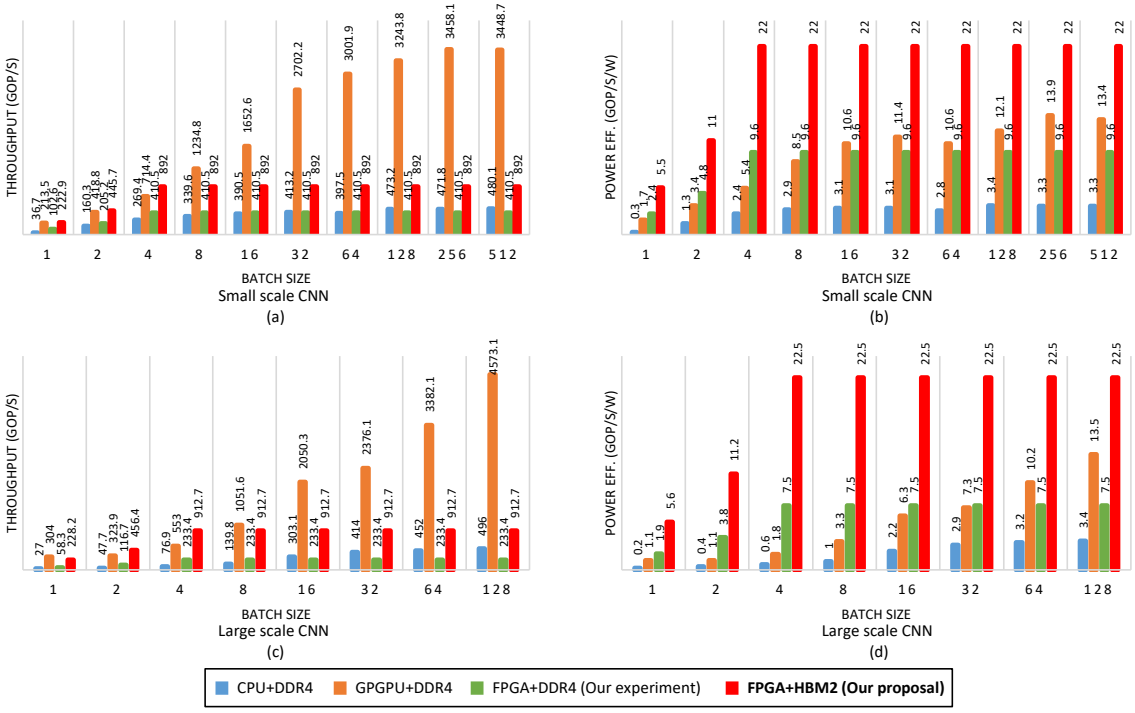

Figure 19. Throughput and power efficiency comparison with DDR-based CPU/ GPGPU.

Fig. 19(a) and 19(c) show that from a batch size of 8 to 128/512, the throughput of the GPGPU gradually increases, while our system remains constant. At a batch size of 128/256, GPGPU reaches its best throughput (4,573/3,458), which is 5×/3.9× higher than our proposed system with the large/small-scale models at the batch size of 4.

However, the multi-core parallel execution with a large number of batch sizes makes the GPGPU consume a huge amount of power. At its peak throughput, the GPGPU consumes up to 249W/311W for small/large-scale models, which leads to poor power efficiency. The graphs in Fig. 19(b) and 19(d) demonstrates the superiority of the proposed system in terms of power efficiency, outperforming the GPGPU, even with various batch sizes. The power efficiency of the proposed system reaches 5.5/5.6 GOP/s/W and 11/11.2 GOP/s/W for small/large-scale models at the batch sizes of 1 and 2, respectively. Furthermore, it achieves a maximum of 22.48 GOP/s/W at the batch size of 4. Compared with the CPU,

our system is roughly 6.5-21.2×/6.6-37.1× better with small/large-scale models. Compared with the GPGPU, our system achieves approximately 1.6-4.1×/1.7-12.6× better with small/large-scale models. The GPGPU achieves its best power efficiency at a batch size of 128/256 (13.5/13.9 GOP/s/W) with large/small scale models, which is still 1.7 × /1.6× lower than that of our system.

6 Conclusion

In this study, an FPGA-based CNN inference accelerator is implemented at the system-on-chip level, and combined with HBM2 as off-chip memory based on Verilog HDL instead of high-level synthesis as the previous researchs. The proposed accelerator has a multi-core architecture, using shared-dual buffers to reduce off-chip memory access and maximize the throughput. Each core works in a fully-pipeline manner, which consists of inter-layer pipeline architecture and pipelined computation engines. For evaluation, the proposed system explores the small/large-scale VGG-16 models with CIFAR100/ImageNet dataset on the Xilinx Alveo U280 card. As the experimental results, the proposed accelerator reaches 912.7 GOP/s and 22.48 GOP/s/W in peak throughput and power efficiency with large scale CNN model, respectively. In throughput, our proposed system is better 6.7× (FPGA+DDR based [84]), 3× (FPGA+DDR based [85]), 1.9× (FPGA+HBM2 based [151]) with large scale model. Moreover, our accelerator is better 1.3×/1.65× (DDR+GPGPU based) and 3.3×/11.9× (DDR+CPU based) in throughput with the small/large-scale models at low batch size (4). In terms of power efficiency, compared with the previous DDR+FPGA/ DDR+GPGPU/ DDR+CPU based accelerators, our proposed system provides 1.4-1.7×/1.7-12.6×/6.6-37.1× improvement with the large-scale CNN model.

Acknowledgements

First of all, I would like to express my sincere thankfulness to Professor Yasuhiko Nakashima for his ongoing and enthusiastic supervision during the three years of my doctoral course. His kind advice and guidance helped me realize my weaknesses and improve myself to become a better student and researcher. He not only patiently explains step-by-step to complete my proposal, but also shares his high-professional knowledge and helps me have a large-and-far vision of the computing architecture field. Besides, I would like to thank Professor Yuichi Hayashi, Associate Professor Renyuan Zhang, Assistant Professor Kan Yirong, and Visiting Associate Professor Tran Thi Hong for serving on my thesis committee and giving valuable suggestions.

Furthermore, I would like to thank the Computing Architecture Laboratory, Division of Information Science, Nara Institute of Science and Technology (NAIST) for supporting my academic career. I feel so lucky to get a chance to work with a lot of wonderful colleagues at the Computing Architecture Laboratory in the last three years. Additionally, I feel happy when spending my life with a small Vietnamese community at NAIST during that time. I would like to thank all people who aid me to overcome the difficult and boring life during the Covid-19 pandemic in Japan. I also would like to thank NAIST for giving me the scholarship to study in Japan.

Finally, my sincere gratitude to my big family in Vietnam for their encouragement and immense support during my doctorate course.

References

[1] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. *Advances in neural information processing systems*, 26, 2013.

[2] Mohamed Abd Elaziz, Abdelghani Dahou, Laith Abualigah, Liyang Yu, Mohammad Alshinwan, Ahmad M Khasawneh, and Songfeng Lu. Advanced metaheuristic optimization techniques in applications of deep neural networks: a review. *Neural Computing and Applications*, 33(21):14079–14099, 2021.

[3] Sajjad Nematzadeh, Farzad Kiani, Mahsa Torkamanian-Afshar, and Nizamettin Aydin. Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases. *Computational Biology and Chemistry*, 97:107619, 2022.

[4] Mohamed A Abdou. Literature review: Efficient deep neural networks techniques for medical image analysis. *Neural Computing and Applications*, pages 1–22, 2022.

[5] Eric J Snider, Sofia I Hernandez-Torres, and Emily N Boice. An image classification deep-learning algorithm for shrapnel detection from ultrasound images. *Scientific reports*, 12(1):1–12, 2022.

[6] Jordan J Bird, Chloe M Barnes, Luis J Manso, Anikó Ekárt, and Diego R Faria. Fruit quality and defect image classification with conditional gan data augmentation. *Scientia Horticulturae*, 293:110684, 2022.

[7] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9):3840–3854, 2020.

[8] Mahbub Hussain, Jordan J Bird, and Diego R Faria. A study on cnn transfer learning for image classification. In *UK Workshop on computational Intelligence*, pages 191–202. Springer, 2018.

[9] Mengmeng Zhang, Wei Li, and Qian Du. Diverse region-based cnn for hyperspectral image classification. *IEEE Transactions on Image Processing*, 27(6):2623–2634, 2018.

[10] Milan Tripathi. Analysis of convolutional neural network based image classification techniques. *Journal of Innovative Image Processing (JIIP)*, 3(02):100–117, 2021.

[11] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia computer science*, 132:377–384, 2018.

[12] Ce Zhang, Xin Pan, Huapeng Li, Andy Gardiner, Isabel Sargent, Jonathon Hare, and Peter M Atkinson. A hybrid mlp-cnn classifier for very fine resolution remotely sensed image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:133–144, 2018.

[13] Xiangteng He and Yuxin Peng. Fine-grained image classification via combining vision and language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5994–6002, 2017.

[14] D Jude Hemanth, J Anitha, Antoanela Naaji, Oana Geman, Daniela Elena Popescu, et al. A modified deep convolutional neural network for abnormal brain image classification. *IEEE Access*, 7:4275–4283, 2018.

[15] Shin-Jye Lee, Tonglin Chen, Lun Yu, and Chin-Hui Lai. Image classification based on the boost convolutional neural network. *IEEE Access*, 6:12755–12768, 2018.

[16] Juan M Gandarias, Alfonso J Garcia-Cerezo, and Jesus M Gomez-de Gabriel. Cnn-based methods for object recognition with high-resolution tactile sensors. *IEEE Sensors Journal*, 19(16):6872–6882, 2019.

[17] Keiji Yanai, Ryosuke Tanno, and Koichi Okamoto. Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 362–366, 2016.

[18] Kripasindhu Sarkar, Kiran Varanasi, Didier Stricker, K Sarkar, K Varanasi, and D Stricker. Trained 3d models for cnn based object recognition. In *VISIGRAPP (5: VISAPP)*, pages 130–137, 2017.

[19] Rohan Ghosh, Abhishek Mishra, Garrick Orchard, and Nitish V Thakor. Real-time object recognition and orientation estimation using an event-based camera and cnn. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 544–547. IEEE, 2014.

[20] Max Schwarz, Hannes Schulz, and Sven Behnke. Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 1329–1335. IEEE, 2015.

[21] Ali Caglayan and Ahmet Burak Can. Exploiting multi-layer features using a cnn-rnn approach for rgb-d object recognition. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[22] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.

[23] Anran Wang, Jiwen Lu, Jianfei Cai, Tat-Jen Cham, and Gang Wang. Large-margin multi-modal deep learning for rgb-d object recognition. *IEEE Transactions on Multimedia*, 17(11):1887–1898, 2015.

[24] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687. IEEE, 2015.

[25] Jawadul H Bappy and Amit K Roy-Chowdhury. Cnn based region proposals for efficient object detection. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3658–3662. IEEE, 2016.

[26] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS research*, 43(4):244–252, 2019.

[27] Han Zhang, Tao Xu, Mohamed Elhoseiny, Xiaolei Huang, Shaoting Zhang, Ahmed Elgammal, and Dimitris Metaxas. Spda-cnn: Unifying semantic part detection and abstraction for fine-grained recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1143–1152, 2016.

[28] Yajie Miao, Mohammad Gowayyed, and Florian Metze. Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 167–174. IEEE, 2015.

[29] Jinyu Li, Rui Zhao, Hu Hu, and Yifan Gong. Improving rnn transducer modeling for end-to-end speech recognition. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 114–121. IEEE, 2019.

[30] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[31] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[32] Tan Lee, Pak-Chung Ching, and Lai-Wan Chan. Recurrent neural networks for speech modeling and speech recognition. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3319–3322. IEEE, 1995.

[33] Bin Zhao, Xuelong Li, and Xiaoqiang Lu. Cam-rnn: Co-attention model based rnn for video captioning. *IEEE Transactions on Image Processing*, 28(11):5552–5565, 2019.

[34] Ning Xu, An-An Liu, Yongkang Wong, Yongdong Zhang, Weizhi Nie, Yuting Su, and Mohan Kankanhalli. Dual-stream recurrent neural network for video captioning. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(8):2482–2493, 2018.

[35] Yingwei Pan, Ting Yao, Houqiang Li, and Tao Mei. Video captioning with transferred semantic attributes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6504–6512, 2017.

[36] Yin Fan, Xiangju Lu, Dian Li, and Yuanliu Liu. Video-based emotion recognition using cnn-rnn and c3d hybrid networks. In *Proceedings of the 18th ACM international conference on multimodal interaction*, pages 445–450, 2016.

[37] Lianli Gao, Zhao Guo, Hanwang Zhang, Xing Xu, and Heng Tao Shen. Video captioning with attention-based lstm and semantic consistency. *IEEE Transactions on Multimedia*, 19(9):2045–2055, 2017.

[38] Weili Fang, Peter ED Love, Hanbin Luo, and Lieyun Ding. Computer vision for behaviour-based safety in construction: A review and future directions. *Advanced Engineering Informatics*, 43:100980, 2020.

[39] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[40] Neena Aloysius and M Geetha. A review on deep convolutional neural networks. In *2017 international conference on communication and signal processing (ICCSP)*, pages 0588–0592. IEEE, 2017.

[41] Elizabeth A Holm, Ryan Cohn, Nan Gao, Andrew R Kitahara, Thomas P Matson, Bo Lei, and Srujana Rao Yarasi. Overview: Computer vision and machine learning for microstructural characterization and analysis. *Metallurgical and Materials Transactions A*, 51(12):5985–5999, 2020.

[42] Yang-Jie Cao, Li-Li Jia, Yong-Xia Chen, Nan Lin, Cong Yang, Bo Zhang, Zhi Liu, Xue-Xiang Li, and Hong-Hua Dai. Recent advances of generative adversarial networks in computer vision. *IEEE Access*, 7:14985–15006, 2018.

[43] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[44] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[46] Dimitrios Marmanis, Mihai Datcu, Thomas Esch, and Uwe Stilla. Deep learning earth observation classification using imagenet pretrained networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):105–109, 2015.

[47] Marcel Simon, Erik Rodner, and Joachim Denzler. Imagenet pre-trained models with batch normalization. *arXiv preprint arXiv:1612.01452*, 2016.

[48] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[49] Leon Yao and John Miller. Tiny imagenet classification with convolutional neural networks. *CS 231N*, 2(5):8, 2015.

[50] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. [dl] a survey of fpga-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(1):1–26, 2019.

[51] Wilson WL Fung, Ivan Sham, George Yuan, and Tor M Aamodt. Dynamic warp formation and scheduling for efficient gpu control flow. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 407–420. IEEE, 2007.

[52] Minsoo Rhu and Mattan Erez. Maximizing simd resource utilization in gpgpus with simd lane permutation. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 356–367, 2013.

[53] Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239. IEEE, 2012.

[54] Mario Vestias and Horacio Neto. Trends of cpu, gpu and fpga for high-performance computing. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6. IEEE, 2014.

[55] Sparsh Mittal and Jeffrey S Vetter. A survey of methods for analyzing and improving gpu energy efficiency. *ACM Computing Surveys (CSUR)*, 47(2):1–23, 2014.

[56] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees Vissers, Joseph Zambreno, and Phillip H Jones. Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In *2019 IEEE international conference on embedded software and systems (ICESS)*, pages 1–8. IEEE, 2019.

[57] Tan Nguyen, Samuel Williams, Marco Siracusa, Colin MacLean, Douglas Doerfler, and Nicholas J Wright. The performance and energy efficiency potential of fpgas in scientific computing. In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 8–19. IEEE, 2020.

[58] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 26(2):203–215, 2007.

[59] Amara Amara, Frederic Amiel, and Thomas Ea. Fpga vs. asic for low power applications. *Microelectronics journal*, 37(8):669–677, 2006.

[60] Andrew Boutros, Sadegh Yazdanshenas, and Vaughn Betz. You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–23, 2018.

[61] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Gee Hock Ong, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, and Guy Boudoukh. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? *2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 5–14, 2017.

[62] Intel Inc. Intel® agilex™ fpga agility and flexibility for the data-centric world. *Product brief*, pages 3–4, 2019.

[63] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 152–159. IEEE, 2017.

[64] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9. IEEE, 2016.

[65] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84, 2017.

[66] Abhinav Podili, Chi Zhang, and Viktor Prasanna. Fast and efficient implementation of convolutional neural networks on fpga. In *2017 IEEE 28Th international conference on application-specific systems, architectures and processors (ASAP)*, pages 11–18. IEEE, 2017.

[67] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24, 2017.

[68] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[69] Kang Ho Lee and Sung Ho Bae. Compressing Neural Networks with Inter Prediction and Linear Transformation. *IEEE Access*, 9:69601–69608, 2021.

[70] Wang Zhe, Jie Lin, Mohamed Sabry Aly, Sean Young, Vijay Chandrasekhar, and Bernd Girod. Rate-Distortion Optimized Coding for Efficient CNN Compression. *Data Compression Conference Proceedings*, pages 253–262, 2021.

[71] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[72] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. FracBNN: Accurate and fpga-efficient binary neural networks with fractional activations. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 171–182, 2021.

[73] Junbin Wang, Shaoxia Fang, Xi Wang, Jiangsha Ma, Taobo Wang, and Yi Shan. High-Performance Mixed-Low-Precision CNN Inference Accelerator on FPGA. *IEEE Micro*, 41:31–38, 2021.

[74] Sean Young, Zhe Wang, David Taubman, and Bernd Girod. Transform Quantization for CNN Compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15, 2021.

[75] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 26–35, 2016.

[76] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[77] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[78] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. *The Annual International Symposium on Microarchitecture, MICRO*, pages 609–622, 2015.

[79] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting vision processing closer to the sensor. *International Symposium on Computer Architecture*, pages 92–104, 2015.

[80] Jian Meng, Shreyas Kolala Venkataramanaiah, Chuteng Zhou, Patrick Hansen, Paul Whatmough, and Jae-sun Seo. FixyFPGA: Efficient FPGA Accelerator for Deep Neural Networks with High Element-Wise Sparsity and without External Memory Access. pages 9–16, 2021.

[81] Xilinx Inc. Adaptable Accelerator Cards for Data Center Workloads. pages 1–2, 2019.

[82] Hongjing Huang, Zeke Wang, Jie Zhang, Zhenhao He, Chao Wu, Jun Xiao, and Gustavo Alonso. Shuhai: A Tool for Benchmarking HighBandwidth Memory on FPGAs. *IEEE Transactions on Computers*, 9340:1–12, 2021.

[83] Xilinx Inc. Supercharge Your AI and Database Applications with Xilinx ' s HBM-Enabled UltraScale + Devices Featuring Samsung HBM2. 508:1–13, 2019.

[84] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37:35–47, 2018.

[85] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, and Yun Liang. An efficient hardware accelerator for sparse convolutional neural networks on FPGAS. *27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, pages 17–25, 2019.

[86] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae Sun Seo. Automatic Compilation of Diverse CNNs Onto High-Performance FPGA Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:424–437, 2020.

[87] J Stuart et al. Artificial intelligence a modern approach third edition, 2010.

[88] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[89] Xue-Bo Jin, Ting-Li Su, Jian-Lei Kong, Yu-Ting Bai, Bei-Bei Miao, and Chao Dou. State-of-the-art mobile intelligence: Enabling robots to move like humans by estimating mobility with artificial intelligence. *Applied Sciences*, 8(3), 2018.

[90] Thomas Ross. Machines that think. *Scientific American*, 148(4):206–208, 1933.

[91] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[92] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[93] Nicole Rusk. Deep learning. *Nature Methods*, 13(1):35–35, 2016.

[94] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[95] Y Bengio. Learning deep architectures for ai. found trends mach learn 2: 1–127, 2009.

[96] Robert Dale, Hermann Moisl, and Harold Somers. *Handbook of natural language processing*. CRC press, 2000.

[97] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[98] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[99] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Rajiv Suman. Artificial intelligence applications for industry 4.0: A literature-based study. *Journal of Industrial Integration and Management*, 7(01):83–111, 2022.

[100] Nils J Nilsson. *The quest for artificial intelligence*. Cambridge University Press, 2009.

[101] Yunhe Pan. Heading toward artificial intelligence 2.0. *Engineering*, 2(4):409–413, 2016.

[102] Seong Ho Park and Kyunghwa Han. Methodologic guide for evaluating clinical performance and effect of artificial intelligence technology for medical diagnosis and prediction. *Radiology*, 286(3):800–809, 2018.

[103] Pavel Hamet and Johanne Tremblay. Artificial intelligence in medicine. *Metabolism*, 69:S36–S40, 2017.

[104] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

[105] Pedro Larranaga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, Inaki Inza, José A Lozano, Rubén Armananzas, Guzmán Santafé, Aritz Pérez, et al. Machine learning in bioinformatics. *Briefings in bioinformatics*, 7(1):86–112, 2006.

[106] Andreas D Baxevanis, Gary D Bader, and David S Wishart. *Bioinformatics*. John Wiley & Sons, 2020.

[107] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560(7716):41–48, 2018.

[108] Sheena Angra and Sachin Ahuja. Machine learning and its applications: A review. In *2017 international conference on big data analytics and computational intelligence (ICBDAC)*, pages 57–60. IEEE, 2017.

[109] Ajay Agrawal, Joshua Gans, and Avi Goldfarb. *Prediction machines: the simple economics of artificial intelligence*. Harvard Business Press, 2018.

[110] Narendra Rao Tadapaneni. Artificial intelligence in finance and investments. *International journal of innovative research in science, engineering and technology*, 9(5), 2019.

[111] Thomas F Gordon. *The pleadings game: An artificial intelligence model of procedural justice*. Springer Science & Business Media, 2013.

[112] Keshav Rathi, Priyam Somani, Aditya V Koul, and KS Manu. Applications of artificial intelligence in the game of football: The global perspective. *Researchers World*, 11(2):18–29, 2020.

[113] Michael Brady. Artificial intelligence and robotics. In *Robotics and Artificial Intelligence*, pages 47–63. Springer, 1984.

[114] Joseph E Aoun. *Robot-proof: higher education in the age of artificial intelligence*. MIT press, 2017.

[115] S Agatonovic-Kustrin and Rosemary Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

[116] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Abubakar Malah Umar, Okafor Uchenwa Linus, Humaira Arshad, Abdullahi Aminu Kazaure, Usman Gana, and Muhammad Ubale Kiru. Comprehensive review of artificial neural network applications to pattern recognition. *IEEE Access*, 7:158820–158846, 2019.

[117] Najmeh Nazari, Mohammad Loni, Mostafa E Salehi, Masoud Daneshtalab, and Mikael Sjodin. Tot-net: An endeavor toward optimizing ternary neural networks. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 305–312. IEEE, 2019.

[118] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. Deep retinal image understanding. In *International conference on medical image computing and computer-assisted intervention*, pages 140–148. Springer, 2016.

[119] Gencer Sumbul, Marcela Charfuelan, Begüm Demir, and Volker Markl. Bigearthnet: A large-scale benchmark archive for remote sensing image understanding. In *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 5901–5904. IEEE, 2019.

[120] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[121] Dimitri Palaz, Ronan Collobert, et al. Analysis of cnn-based speech recognition system using raw speech as input. Technical report, Idiap, 2015.

[122] Nan Zhang, Jianzong Wang, Wenqi Wei, Xiaoyang Qu, Ning Cheng, and Jing Xiao. Cacnet: Cube attentional cnn for automatic speech recognition. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2021.

[123] Luca Comanducci, Paolo Bestagini, Marco Tagliasacchi, Augusto Sarti, and Stefano Tubaro. Reconstructing speech from cnn embeddings. *IEEE Signal Processing Letters*, 28:952–956, 2021.

[124] Berat Yildiz, Akif Durdu, AHMET KAYABAŞI, and Mehmet Duramaz. Cnn based sensor fusion method for real-time autonomous robotics systems. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(1):79–93, 2022.

[125] Yuxin Liu, Chong Chen, Tao Wang, and Lianglun Cheng. An attention enhanced dilated cnn approach for cross-axis industrial robotics fault diagnosis. *Autonomous Intelligent Systems*, 2(1):1–11, 2022.

[126] Paolo Arena, Luigi Fortuna, Mattia Frasca, and Luca Patané. A cnn-based chip for robot locomotion control. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(9):1862–1871, 2005.

[127] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231, 2018.

[128] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. Accelerating cnn inference on fpgas: A survey. *arXiv preprint arXiv:1806.01683*, 2018.

[129] Kyungjun Cho, Hyunsuk Lee, Heegon Kim, Sumin Choi, Youngwoo Kim, Jaemin Lim, Joungho Kim, Hyungsoo Kim, Yongju Kim, and Yunsaing Kim. Design optimization of high bandwidth memory (hbm) interposer considering signal integrity. In *2015 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, pages 15–18. IEEE, 2015.

[130] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. Fine-grained dram: Energy-efficient dram for extreme bandwidth systems. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 41–54. IEEE, 2017.

[131] Joe Macri. Amd's next generation gpu and high bandwidth memory architecture: Fury. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–26. IEEE, 2015.

[132] Mike Wissolik, Darren Zacher, Anthony Torza, and Brandon Da. Virtex ultrascale+ hbm fpga: a revolutionary increase in memory performance. *Xilinx Whitepaper*, 2017.

[133] Bo Pu. Design of 2.5 d interposer in high bandwidth memory and through silicon via for high speed signal. 2020.

[134] Ultrascale, Xilinx Inc. Supercharge Your AI and Database Applications with Xilinx ' s HBM-Enabled UltraScale + Devices Featuring Samsung HBM2. 508:1–13, 2019.

[135] Xilinx Inc. AXI High Bandwidth. 276, 2019.

[136] Philipp Holzinger, Daniel Reiser, Tobias Hahn, and Marc Reichenbach. Fast hbm access with fpgas: Analysis, architectures, and applications. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 152–159. IEEE, 2021.

[137] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[138] Shuai Li, Yukui Luo, Kuangyuan Sun, Nandakishor Yadav, and Kyuwon Ken Choi. A novel fpga accelerator design for real-time and ultra-low power deep convolutional neural networks compared with titan x gpu. *IEEE Access*, 8:105455–105471, 2020.

[139] Li Yang, Zhezhi He, and Deliang Fan. A fully onchip binarized convolutional neural network fpga impelmentation with accurate inference. In

Proceedings of the International Symposium on Low Power Electronics and Design, pages 1–6, 2018.

[140] Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot, Hande Alemdar, Nicholas Caldwell, and Vincent Leroy. Scalable high-performance architecture for convolutional ternary neural networks on fpga. In *2017 27Th International conference on field programmable logic and applications (FPL)*, pages 1–7. IEEE, 2017.

[141] Adrien Prost-Boucle, Alban Bourge, and Frédéric Pétrot. High-efficiency convolutional ternary neural networks with custom adder trees and weight compression. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–24, 2018.

[142] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rebnet: Residual binarized neural network. In *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*, pages 57–64. IEEE, 2018.

[143] Li Jiao, Cheng Luo, Wei Cao, Xuegong Zhou, and Lingli Wang. Accelerating low bit-width convolutional neural networks with embedded fpga. In *2017 27th international conference on field programmable logic and applications (FPL)*, pages 1–4. IEEE, 2017.

[144] Yixing Li, Zichuan Liu, Kai Xu, Hao Yu, and Fengbo Ren. A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks. In *FPGA*, pages 290–291, 2017.

[145] Hiroki Nakahara, Haruyoshi Yonekawa, Hisashi Iwamoto, and Masato Motomura. A batch normalization free binarized convolutional deep neural network on an fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 290–290, 2017.

[146] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[147] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[148] Jian Meng, Shreyas Kolala Venkataramanaiah, Chuteng Zhou, Patrick Hansen, Paul Whatmough, and Jae-sun Seo. Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 9–16. IEEE, 2021.

[149] Lei Gong, Chao Wang, Xi Li, Huaping Chen, and Xuehai Zhou. MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37:2601–2612, 2018.

[150] Libo Chang, Shengbing Zhang, Huimin Du, Yue Chen, and Shiyu Wang. A Reconfigurable Neural Network Processor With Tile-Grained Multicore Pipeline for Object Detection on FPGA. *IEEE Transactions on Very Large Scale Integration Systems*, 29:1967–1980, 2021.

[151] Ryosuke Kuramochi and Hiroki Nakahara. An FPGA-Based Low-Latency Accelerator for Randomly Wired Neural Networks. *30th International Conference on Field-Programmable Logic and Applications*, pages 298–303, 2020.

[152] Zhiqiang Liu, Yong Dou, Jingfei Jiang, Jinwei Xu, Shijie Li, Yongmei Zhou, and Yingnan Xu. Throughput-optimized FPGA accelerator for deep convolutional neural networks. *ACM Transactions on Reconfigurable Technology and Systems*, 10, 2017.

[153] [Online]. The cifar-100 dataset. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[154] Xu Jinwei and et al. Cafpga: An automatic generation model for cnn accelerator. In *Elsevier Journal of Microprocessors and Microsystems*, volume 60, pages 196–206, 2018.

List of Publication

Peer review journal paper

1. Van-Cam NGUYEN, and Yasuhiko NAKASHIMA, "Implementation of Fully-Pipelined CNN Inference Accelerator on FPGA and HBM2 Platform", IEICE Transactions on Information and Systems, Vol.E106-D, No.6, pp.1117-1129, Jun. (2023). (Sections 1.1, 2.3, 2.4, 2.5, 3, 4, 5, and 6)

Peer review workshops paper

1. Van-Cam NGUYEN, and Yasuhiko NAKASHIMA, "Analysis of Fully-Pipelined CNN Implementation on FPGA and HBM2", 2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW), poster, pages 134–137, 2021. (Sections 2.4, and 3)