A High-performance CNN Processor Based on FPGA for MobileNets

Di Wu, Yu Zhang, Xijie Jia, Lu Tian, Tianping Li, Lingzhi Sui, Dongliang Xie, Yi Shan
1Xilinx Inc., 2Tsinghua University
{diw, ayzhang, jasonjia, lutian, tianping, lingzhi, dlxie, yishan}@xilinx.com

Abstract—Convolution neural networks (CNNs) have been widely applied in the fields of computer vision tasks. However, it is hard to deploy those standard neural networks into embedded devices because of their large amount of operations and parameters. MobileNet, the state-of-the-art CNN which adopts depthwise separable convolution to replace the standard convolution has significantly reduced operations and parameters with only limited loss in accuracy. A high-performance CNN processor based on FPGA is proposed in this paper. To improve the efficiency, two dedicated computing engines named Conv Engine and Dwcw Engine were designed for pointwise convolution and depthwise convolution respectively. The schedule for Conv Engine and Dwcw Engine has significantly improved the efficiency of our accelerator. Furthermore, we designed a special architecture called Channel Augmentation to improve the efficiency in the first layer of MobileNets. The accelerator can be flexibly deployed to various devices with different configurations to balance hardware resources and computational performance. We implemented our accelerator on ZU2 and ZU9 MPSoC FPGAs. The classification on ImageNet achieved 205.3 frames per second (fps) on ZU2 and 809.8 fps on ZU9, which is 15.4x speedup on ZU2 and 60.7x speedup on ZU9 compared to CPU. We also deployed MobileNet + SSD network on our accelerator for object detection, and achieved 31.0 fps on ZU2 and 124.3 fps on ZU9.

Index Terms—convolution neural network, FPGA, hardware accelerator, MobileNet

I. INTRODUCTION

Nowadays, the convolution neural networks (CNNs) have been widely applied in the fields of image classification [20], object detection [23] and semantic segmentation [24]. The state-of-the-art CNNs are having more layers and higher computation complexity which make it hard to be deployed into embedded devices. Using FPGA to accelerate CNN has attracted significant attention in the field of Data Center acceleration [27] and Automotive Driver Assistance (ADAS). FPGA has the ability to support the rapidly changing CNN models due to its flexibility and ease of development.

The standard CNNs have been trending towards more layers, more complex structures, and more complicated operations in pursuit of high accuracy [12, 13, 14, 15]. The massive amount of operations and parameters make a strict requirement in memory throughput and computation capability. A novel convolution architecture was firstly proposed in [17] to lighten the computing burden of standard convolutions and to decrease the amount of parameters. The state-of-the-art CNNs, such as Xception [21], MobileNetV2 [1] and ShuffleNet [9] adopt depthwise separable convolution to replace the standard convolution, which significantly reduced operations and parameters with only limited loss in accuracy.

CNN is a computing intensive task which consumes huge amounts of computing power. The Graphic Processing Units (GPUs) are used to be selected as the target platform due to its adequate performance. However, the power consumption poses a serious challenge to GPUs. As a result, FPGAs have seen a surge in interest for CNN acceleration due to their programmable, massive parallel and power-efficient computing substrate [3].

Despite recent efforts to use FPGAs to accelerate CNNs, there still exists a wide gap between accelerator design and CNN model [16]. Some FPGA accelerators are still targeted at large and inefficient standard CNN models such as AlexNet [18], VGG16 [12], GoogLeNet [4] and ResNet [19]. Those inefficient models require greater storage and computational resources than the small and efficient models achieving same accuracy. Accelerator based on these inefficient models are hard to obtain a high performance.

Recent accelerators based on the state-of-the-art CNNs such as MobileNets, ShuffleNets only achieved acceptable speed on image recognition tasks. Those novel CNNs usually utilize depthwise separable convolution to reduce operations and parameters. In [8], a CNN accelerator for MobileNetV2 reached a processing speed of 266.2 fps on Arria 10 SoC. Another accelerator based on RR-MobileNet achieved 64.6% Top1 accuracy and 84.5% Top5 accuracy on ImageNet classification, and achieved 127.4 fps on Xilinx ZU9EG [5]. [7] proposed an accelerator named Syntegy for DiracDeltaNet which is an optimized model based on ShuffleNetV2. Syntegy achieved 96.5 fps on ZU3EG platform, and the top-1 accuracy on ImageNet classification is 68.47%. However, all these accelerators are imbalanced in computational resources and on-chip memory resources, which limits their performance. One reason is that utilizing one computing common engine to accelerate both standard convolution and depthwise convolution. In this paper, we proposed a high-performance accelerator based on FPGA for MobileNets. The MobileNets are quantized with 8-bit fixed point, and achieved top-1 68.1% accuracy on ImageNet classification. Our accelerator can be flexibly deployed to different FPGAs with scalable configurations, achieving a high performance while balancing on-chip resources.

The key contributions of this work are:
- A novel CNN hardware accelerator architecture is pro-
posed which consists of a dedicated convolution engine and a depthwise convolution engine.

- Efficiently scheduling the convolution engine and the depthwise convolution engine allows the two to run in parallel through layer pipeline.
- An optimization strategy named Channel Augmentation which improves efficiency of the accelerator for the first layer in MobileNets.
- A methodology for scalable design which can be implemented in various FPGAs by balancing the FPGA resource utilization and performance.

This paper is organized as follows. Section II introduces the theory of depthwise separable convolution and its following application MobileNets. Section III describes the proposed architecture design and acceleration. The system implementation and experiment results are discussed in Section IV. The conclusion is given in Section V.

## II. BACKGROUND

### A. Depthwise separable convolution

The depthwise separable convolution (DSC) is a form of factorized standard convolution (SC) which was proposed in [21]. Fig.1 demonstrates SC and DSC. In a SC, each input channel does convolution with its specific kernel, the convolution results of all input channels are summed to produce one output channel. There are \( N \) sets of kernels for \( N \) output channels. In DSC, the SC is split into depthwise convolution (Dwcv) and pointwise convolution (Pwcv). The Dwcv applies kernel to each input channel individually to produce the same number of output channels. Pwcv is actually SC with \( 1 \times 1 \) kernel. The factorization of SC has greatly reduced the number of operations and parameters. As shown in Fig.1, the size of input feature map is considered as \( W \times W \times M \), the kernel size is \( K \times K \) and the output feature size is considered as \( W \times W \times N \), \( W \) is the spatial width and height of a square input feature map, \( M \) is the number of input channels, \( N \) is the number of output channels. In case of stride length of 1, the amount of operations is computed as:

\[
O_{SC} = W \times W \times K \times K \times M \times N \tag{1}
\]

In case of DSC, the amount of operations is:

\[
O_{DSC} = W \times W \times K \times K \times M + W \times W \times M \times N \tag{2}
\]

which is the sum of Dwcv and Pwcv.

By splitting a SC into a Dwcv and a Pwcv we get a reduction operations of:

\[
F_O = \frac{O_{DSC}}{O_{SC}} = \frac{1}{N} + \frac{1}{K^2} \tag{3}
\]

In fact, the reduction of parameters is also \( \frac{1}{N} + \frac{1}{K^2} \).

### B. MobileNetV1 and MobileNetV2

MobileNets are based on a streamlined architecture that uses DSCs to build light weight deep neural networks[10]. MobileNetV1 can achieve a 3% higher top-1 accuracy than AlexNet[18], but its size is 45x smaller. MobileNetV2, the successor of MobileNetV1, has further decreased the number of weights and improved accuracy. It applies an inverted residual structure between the thin bottleneck layers. The imported linear bottleneck improves the performance and shrinks the output channels in some layers. The convolution blocks in MobileNetV2 is shown in Fig.2. The non-linearity is replaced with ReLU6 because of its robustness. MobileNetV2 achieves 72% top-1 accuracy on ImageNet classification with a parameter size of 3.4M[1]. MobileNets can also be applied in the object detection framework such as SSD[26], and MobileNetV1 + SSD achieves 19.3% mAP on COCO dataset [11] with few parameters.

MobileNets with small model size have a competitive accuracy both on classification and object detection. This is a significant advantage for MobileNets to process computer vision tasks on the edge devices.

---

**Fig. 1.** Standard convolution and Depthwise separable convolution

**Fig. 2.** Convolution blocks in MobileNetV2
TABLE I

<table>
<thead>
<tr>
<th>Notation</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_h$</td>
<td>variable</td>
<td>height of the kernel size</td>
</tr>
<tr>
<td>$K_w$</td>
<td>variable</td>
<td>width of the kernel size</td>
</tr>
<tr>
<td>$HI$</td>
<td>variable</td>
<td>height of the input image</td>
</tr>
<tr>
<td>$IW$</td>
<td>variable</td>
<td>width of the input image</td>
</tr>
<tr>
<td>$IC$</td>
<td>variable</td>
<td>total input channel</td>
</tr>
<tr>
<td>$OC$</td>
<td>variable</td>
<td>total output channel</td>
</tr>
<tr>
<td>$CP$</td>
<td>scalable constant</td>
<td>parallelism on channel dimension</td>
</tr>
<tr>
<td>$ICP$</td>
<td>scalable constant</td>
<td>input $CP$</td>
</tr>
<tr>
<td>$OCP$</td>
<td>scalable constant</td>
<td>output $CP$</td>
</tr>
<tr>
<td>$PP$</td>
<td>scalable constant</td>
<td>parallelism on pixel (height) dimension</td>
</tr>
</tbody>
</table>

III. ARCHITECTURE DESIGN AND ACCELERATION

A. Architecture Overview

The proposed high-performance CNN processor is named Deep learning Processing Unit (DPU). The block diagram Fig.3 gives an overview of DPU. Our DPU is driven by instructions which are pre-compiled before run time. In our system, the Processing System (PS) prepares the input image and instructions into the off-chip memory. When we start up DPU, the instr-scheduler fetches instructions from off-chip memory, and then parses and dispatches them to the memory controller and computation engines. The instruction schedules the related module to load feature map, weights and bias from off-chip memory to on-chip memory. The mult-addr operations are processed in parallel in the computation engines including Conv Engine, Dwcv Engine, Pooling Engine, and Elementwise Engine. The processing results are written back into on-chip memory and saved back to off-chip memory.

B. Workload Analysis

In MobileNetV1 model, 95% of the operations are $1 \times 1$ Pwcv (which is a simple standard conv), 1% are $3 \times 3$ standard conv, and the rest are $3 \times 3$ Dwcv. The proportion is displayed in Table II[10]. The successor MobileNetV2 contains 19 residual bottleneck layers which makes the proportion of the $1 \times 1$ Pwcv slightly increased[1]. The workload between standard conv and Dwcv are imbalanced. Meanwhile, the Dwcv applies only one filter makes the Conv Engine be inefficient for that the engine can process $OCP$ filters in parallel. As illustrated in Fig.4(a), lots of PEs are idle and wasted. In summary, it is inefficiency for running all layers of standard conv and Dwcv in one general Conv Engine. Thus we realized the following pipeline schedule and a dedicated Dwcv Engine to accelerate our proposed design for MobileNets.

C. Pipeline Schedule Among Layers

As mentioned in Fig.2, the most common structure in MobileNetV2 is that one $1 \times 1$ Pwcv layer is followed by a $3 \times 3$ Dwcv layer, and then another $1 \times 1$ Pwcv layer. The output results of one layer are the input feature of the following layer. It indicates that the following layer could start calculating once its sliding window ($K_h \times IW$) fills up on the outputs of the current layer. This inspired us to achieve the on-chip pipeline schedule to efficiently process Pwcv layer and Dwcv layer in parallel, and to reduce the off-chip memory access.

To achieve the proposed pipeline processing, an unique Dwcv Engine is designed. The Dwcv Engine utilizes much less computing power than the original Conv Engine, and focuses on the processing of Dwcv layer. The Conv Engine computes the standard conv and Pwcv layer, for that both of these two types of layers have arbitrary $OC$. The detail design will be discussed in Section III-D.

The pipeline schedule combines the three contiguous layers: Conv-Dwcv-Conv into one layer, which is shown in Fig.4(b). The input feature or weights of some layers may be too large to be filled into the on-chip memory, then we only combine Conv-Dwcv layer, and leave the next Conv layer alone, as shown in Fig.4(c). With the on-chip layer-combination, the Conv layer and Dwcv layer could be processed in parallel for higher efficiency, and the reduced off-chip memory access further decreases the run-time of latter-layer.

The schedule controls the data dependency among layers at row-level. The process of each latter layer should wait for its $K_h$ rows of input feature which is also the output feature of its previous layer. The detail of data flow are described in Section III-E.

D. Scalable Computation Engines

Fig.5 illustrates the scalable computing parallelism of Conv Engine and Dwcv Engine. Conv Engine can process in parallel on the three dimensions: $OCP$, $ICP$, and $PP$ (along $IH$ direction), while Dwcv Engine only needs on $ICP$ and $PP$. 

Authorized licensed use limited to: WASHINGTON UNIVERSITY LIBRARIES. Downloaded on January 22,2023 at 04:58:22 UTC from IEEE Xplore. Restrictions apply.
CONV

DWCV

wasted
time
compute
parallelism
(a) Single Conv Engine running all types of convolution. Each Conv contains read/write access to off-chip memory.
compute
time
pipeline
combined-3-layers.
(b) Conv and Dwcv Engines running pipeline of combined-3-layers.
compute
time
pipeline
2+1 combination.
(c) Conv and Dwcv Engines running pipeline of 2+1 combination.
Fig. 4. Profiling of different strategies of the proposed parallel schedule. The height and width of the rectangle qualitatively indicate the contrasts of compute parallelism and computing time. Only the first layer in the combination needs read access to off-chip memory, and only the last layer needs write access. The profiling shows that the pipeline schedule of combined-layer is good for accelerating the system.

for Dwcv does not combine the input channels to produce one output channel. The parallelism ratio $R_p$ of Conv Engine to Dwcv Engine is calculated by (4). Typically, we bind $ICP_{Conv} = ICP_{Dwcv}$, and $ICP = OCP = CP$. Thus the $R_p$ could be simplified as (5).

$$R_p = \frac{OCP_{Conv} \cdot ICP_{Conv} \cdot PP_{Conv}}{ICP_{Dwcv} \cdot PP_{Dwcv}}$$

$$R_p = CP \cdot PP_{Conv} \cdot PP_{Dwcv}$$

As mentioned in Section III-B, the operations of Conv is nearly 32 times of that of Dwcv in MobileNets. Thus we tune the $R_p$ as well as 32 to balance the run-time (along $IW$ dimension) between Conv Engine and Dwcv Engine to achieve the profiling illustrated in Fig 4(b). Typically, the larger $CP$ and $PP$ are, the faster the system is. While there is an on-chip resource limit. We tune the $CP$ as 16 and $PP_{Conv}$ as 8, for that the number of $IC$ of the most of layers in MobileNets[10, 1] are multiple of 16, and the height of the most of layers are no more than and close to multiple of 8. For the smaller chip, $CP$ and $PP_{Conv}$ may have to be smaller. The performance results of different configurations will be discussed in Section IV.

The elementwise and pooling operation are processed in Elementwise Engine and Pooling Engine respectively in DPU. The Fully Connected layer is mapped onto Conv engine.

E. Data Flow

The on-chip memory (Block-RAM, Distributed-RAM, or Ultra-RAM) is partitioned into banks which are used as caches of feature, weight and bias data. When the accelerator starts to execute a layer, the feature, weight and bias data are firstly loaded from off-chip memory to their corresponding banks. If the data in banks is enough for an engine to work, the engine reads inputs from banks, does the calculation and writes result back to the banks. Finally the result will be saved from banks to off-chip memory. The schedule module controls the work timing of the engines. Figure. 6 shows the data flow of feature when Conv Engine and Dwcv Engine work in pipeline. In this figure Conv Engine reads its input feature from bank0 and writes its result to bank1, Dwcv engine reads Conv’s result as its input from bank1 and writes its result to the same bank. We didn’t instantiate a new bank for Dwcv to store its result because the data throughout of Dwcv is much less than that of Conv. Reuse of bank1 could save the resource of on-chip memory, but when on-chip memory is plenty enough on the device, a new bank could be used to achieve high-performance. The configuration of bank utilization makes our design more scalable.

Fig. 5. Computing parallelism of Conv Engine and Dwcv Engine

Fig. 6. Data flow of feature when Conv and Dwcv work in pipeline
**F. Channel Augmentation**

The convolution operation in our implementation is factorized into three dimensions: \( PP, ICP \) and \( OCP \). The Conv Engine can handle up to \( ICP \) channels for each pixel of \( PP \) in a single cycle. When the \( IC \) is much larger than \( ICP \), it consumes multi-cycles for Conv Engine to complete the processing of total channels. In this situation, the efficiency of Conv Engine on channel dimension is admirable. However, when \( IC \) is less than \( ICP \), the utilization of Conv Engine on input channel dimension is \( IC/ICP \). Especially in the first layer of MobileNets, the \( IC \) is 3, and the maximum efficiency of Conv Engine is 3/16 when the \( ICP \) is set as 16 which is a general configuration in our design. To improve the efficiency of Conv Engine in first layer, a novel data rearrangement pattern named Channel Augmentation is proposed.

In Channel Augmentation, the reused data in feature and weights is rearranged from the input dimension. As shown in Fig.7(a), for a given convolution with \( IC \) of 3, and \( ICP \) of 9, the efficiency of Conv Engine is just 3/9. To utilize the wasted parallelism of Conv Engine, the input feature and weights is transformed from Fig.7(a) to Fig.7(b). In Fig.7(b), the channel dimension of feature and weights is filled with data that should be reused by a sliding convolution kernel in the width direction. Then the \( IC \) is equal with \( ICP \), and the efficiency of Conv Engine is improved from 3/9 to 9/9. The Channel Augmentation has significantly improved the efficiency in the channel dimension of Conv Engine when \( IC \) is less than \( ICP \). The performance results will be discussed in Section IV-B1.

**IV. EXPERIMENTS**

**A. Implementation Notes**

Our CNN accelerator is adopt with several configurable parameters which makes DPU be flexibly deployed on various FPGA devices. The resource utilization of DPU is mainly determined by the scalable parameters \( CP, PP_{Conv}, \) and \( PP_{Dwcv} \). Those parameters directly determine the size of Conv Engine and Dwcv Engine, which occupy most of the resources in DPU.

The quantization strategy is significant for CNN implementation. Though the 32bit model can achieve the best accuracy, a CNN accelerator supporting 32bit quantization will be a waste of resources, hence an inefficient solution. Two 8-bit multiplication can be calculated simultaneously in a single DSP48E2 slice which is the essential computing unit in Xilinx MPSoC with a maximum multiplication of a 18-bit and 27-bit data [28]. For a multiplication of two 32-bit data, four DSP48E2 slices is prerequisite. This means that the 8-bit strategy can achieve 8 times of the computing performance comparing with the 32-bit one with same amounts of DSP48E2. Since the MobileNets is an efficient model without much redundant, the 8-bit quantization for MobileNets appears to be hardly successful. Especially in the depthwise convolution layers, the weights are varied on channel dimension with a large range, which can hardly be quantized in 8-bit without much entropy loss. This compromised quantization method caused a precision degradation in top-1 classification accuracy.

We selected Xilinx XCZU2EG and XCZU9EG MPSoC [29] as the targeted platforms to implement our deep learning...
TABLE III
RESOURCES USAGE

<table>
<thead>
<tr>
<th>Arch</th>
<th>Prec.(w/a)</th>
<th>LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>DSP48E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPU_S(impl)</td>
<td>8b</td>
<td>31198</td>
<td>46809</td>
<td>145</td>
<td>212 β</td>
</tr>
<tr>
<td>DPU_S(synth)</td>
<td>32b</td>
<td>127126</td>
<td>250092</td>
<td>565</td>
<td>1249 β</td>
</tr>
<tr>
<td>DPU_L(impl)</td>
<td>8b</td>
<td>161944</td>
<td>301416</td>
<td>771</td>
<td>2070</td>
</tr>
</tbody>
</table>

α Parts of DSPs are used for accumulation to reduce the LUT resources
β All the DSPs are used for multiplication

TABLE IV
COMPARE BETWEEN W/ AND W/O CHANNEL AUGMENTATION IN THE FIRST LAYER

<table>
<thead>
<tr>
<th>Platform</th>
<th>Configuration</th>
<th>Runtime(ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZU2</td>
<td>w/o augm</td>
<td>0.42</td>
<td>1.91x</td>
</tr>
<tr>
<td></td>
<td>w/ augm</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>ZU9</td>
<td>w/o augm</td>
<td>0.21</td>
<td>1.31x</td>
</tr>
<tr>
<td></td>
<td>w/ augm</td>
<td>0.16</td>
<td></td>
</tr>
</tbody>
</table>

TABLE V
PERFORMANCE COMPARISON IN CLASSIFICATION

<table>
<thead>
<tr>
<th>Design</th>
<th>Network</th>
<th>Platform</th>
<th>Speed (fps)</th>
<th>Top-1 Prec. (W/a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>MobileNetV2</td>
<td>CPU</td>
<td>15.7</td>
<td>72.0</td>
</tr>
<tr>
<td>[5]</td>
<td>RR-MobileNet</td>
<td>ZU9EG</td>
<td>127.4</td>
<td>64.6</td>
</tr>
<tr>
<td>[6]</td>
<td>MobileNetV1</td>
<td>StratixV</td>
<td>231.7</td>
<td>-</td>
</tr>
<tr>
<td>[7]</td>
<td>DiracDeltaNet</td>
<td>ZU3EG</td>
<td>96.5</td>
<td>68.4</td>
</tr>
<tr>
<td>[8]</td>
<td>MobileNetV2</td>
<td>Arria10</td>
<td>260.2</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>MobileNetV2</td>
<td>ZU2EG</td>
<td>205.3</td>
<td>68.1</td>
</tr>
<tr>
<td>Ours</td>
<td>MobileNetV2</td>
<td>ZU9EG</td>
<td>809.8</td>
<td>68.1</td>
</tr>
</tbody>
</table>

TABLE VI
RUNTIME AND FRAME RATE IN DETECTION

<table>
<thead>
<tr>
<th>Framework</th>
<th>Platform</th>
<th>Speed(fps)</th>
<th>Prec.(W/a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV1 + SSD</td>
<td>CPU</td>
<td>3.70</td>
<td>32b</td>
</tr>
<tr>
<td>MobileNetV1 + SSD</td>
<td>ZU2(ours)</td>
<td>31.0</td>
<td>8b</td>
</tr>
<tr>
<td>MobileNetV1 + SSD</td>
<td>ZU9(ours)</td>
<td>124.3</td>
<td>8b</td>
</tr>
</tbody>
</table>

accelerator. ZU2 is usually used as an embedded device with limited logic resources, and ZU9 can be applied in ADAS system with relatively large amount of logic resources. We instantiate a small core named DPU_S on ZU2 board and a large core named DPU_L on ZU9 board. In Table III, we list the implemented resources of DPU_S and DPU_L of 8bit version. For the DPU_S of 32bit version, we have not implemented it, the resources are just after synthesized. Note that both the 8-bit and 32-bit version of DPU_S have the same computing performance.

B. Implementation Results

The DPU runs at 430MHz frequency on the ZU2 board, and runs at 333MHz on the ZU9 board. The ZU2 chip could achieve higher frequency because of its relatively more routing resources and smaller area.

1) Acceleration with Channel Augmentation: We conduct several experiments to figure out the acceleration by Channel Augmentation in DPU. The comparison in the run-time of the first layer is illustrated in Table IV. The speedup of first layer is 1.31x on ZU9 and 1.91x on ZU2. The difference is due to the configurations of the parameter CP, which is 12 on ZU2, and 16 on ZU9. Actually, the first layer is a standard convolution with kernel size 3×3 and IC = 3. As explained in Section III-D, the Channel Augmentation improves the efficiency from 3/12 to 9/12 on ZU2, and 3/16 to 9/16 on ZU9. That’s why the acceleration of Channel Augmentation is more significant on ZU2 than on ZU9.

2) Classification Results on ZU2 and ZU9: Table V provides a comparison between our works and other similar accelerators on ImageNet classification. Our accelerator achieved 205.3 fps on ZU2, which is 2.13 times faster than that of Snyentg on ZU3. Note that there are more logic resources on ZU3 than on ZU2, Though ZU9, Stratix-V and Arria 10 Soc have comparable logic resources, our accelerator’s performance on ZU9 is 3.04 times faster than the best.

3) Detection Results on ZU2 and ZU9: MobileNets can also be applied as an effective base network in modern object detection system [10]. We also implemented DPU for object detection in MobileNetV1 + SSD model on ZU2 and ZU9 FPGAs. There are rare neural network accelerators that provide results for object detection in a MobileNet-like network, so we just compared the performance of object detection between DPU and CPU. As illustrated in Table VI, our accelerator achieves 8.4x speedup on ZU2 and 33.6x speedup on ZU9 compared to CPU. We have only completed the deployment of MobileNetV1 + SSD on the FPGA, and the deployment of MobileNetV2 + SSD is still in progress.

V. CONCLUSION AND FUTURE WORKS

In this paper, a high-performance and scalable accelerator named DPU is proposed. To achieve high efficiency on MobileNets processor, two dedicated convolution calculating engines named Conv Engine and Dwcv Engine are designed in DPU. Furthermore, we used layer pipeline to make Conv Engine and Dwcv Engine working in parallel. A dedicated hardware architecture named Channel Augmentation is proposed to improve the efficiency in the first layer of MobileNets. Our accelerator can be flexibly deployed on FPGAs of different sizes with several configurable parameters. We implemented our accelerator in Xilinx ZU2 and ZU9 FPGA. For image classification, our results achieved 15.4X speedup on ZU2 and 60.7x speedup on ZU9 compared to CPU. For object detection, we implemented DPU as a feature extractor on MobileNetV1 + SSD framework. The object detection results of DPU on ZU2 is 8.4 times faster and 33.6 times faster on ZU9 than that of CPU.

For the future works, we will focus on the further improvement in accuracy and performance. Especially for the quantization strategy, maybe a 10bit or 12bit quantization will perfectly balance the precision and resources.

ACKNOWLEDGEMENT

This work is supported by the Xilinx Deephi department. We would like to thank all the people who helped us realize this project, especially Jinzhang Peng, Xiao Sheng, Yue Gao and Ziheng Gao.
REFERENCES


