2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00010

FixyFPGA: Efficient FPGA Accelerator for Deep Neural Networks with High Element-Wise Sparsity and without External Memory Access

Jian Meng*, Shreyas Kolala Venkataramanaiah*, Chuteng Zhou†, Patrick Hansen†, Paul Whatmough†, and Jae-sun Seo*
*School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA
†Arm ML Research, Boston, MA, USA
Email: jaesun.seo@asu.edu

***Abstract*—Convolutional neural networks (CNNs) have become very popular in real-time computer vision systems. CNNs involve a large amount of computation and storage and typically demand a highly efficient computing platform. Researchers have explored a diverse range of software and hardware optimizations to accelerate CNN inference in recent years. The high power consumption of GPUs and the lack of flexibility with ASIC has promoted interest in FPGAs as a promising platform to efficiently accelerate these CNN inference tasks. Various FPGA-based CNN accelerators have been proposed to low precision weights and high-sparsity in various forms. However, most of the previous work requires off-chip DDR memory to store the parameters and expensive DSP blocks to perform the computation. In this work, we propose the FixyFPGA, a fully on-chip CNN inference accelerator that naturally supports high-sparsity and low-precision computation. In our design, the weights of the trained CNN network are hard-coded into hardware and used as fixed operand for the multiplication. Convolution is performed by streaming the input images to the compute engine in a fully-paralleled, fully-pipelined manner. We analyzed the performance of the proposed scheme with both image classification tasks and object detection tasks based on the low precision, sparse compact CNN models. Compared to prior works, our design achieved 2.34× higher GOPS on ImageNet classification and 3.82× higher frames per second on Pascal VOC object detection.**

***Index Terms*—Convolution neural networks, hardware accelerator, FPGA, low-precision quantization, pruning.**

I. Introduction

Convolutional neural networks (CNNs) have been successful in many practical applications including image classification, object detection and segmentation, and various algorithms and architectures have been proposed in a very fast pace [1]–[4]. GPUs are the *de facto* hardware platform for DNN training workloads, aided by the highly parallel computing with a massive number of processing cores. However, due to the high price and the lack of reconfigurability, GPU is usually not an ideal solution for DNN inference acceleration, especially for models with high sparsity or customized architectures. ASICs such as the Google TPU [5] typically have the highest energy efficiency, but their limited configurability can introduce a significant risk of premature obsolescence, as the model architectures evolve over time. With DNN algorithms evolving at a fast pace, ASIC designs will always lag behind the cutting edge due to the long design cycle. To that end, FPGAs have a unique advantage with potentially higher throughput and efficiency than GPUs, while offering faster time-to-market and potentially longer useful life than ASIC solutions.

This work was in part supported NSF grant 1652866 and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

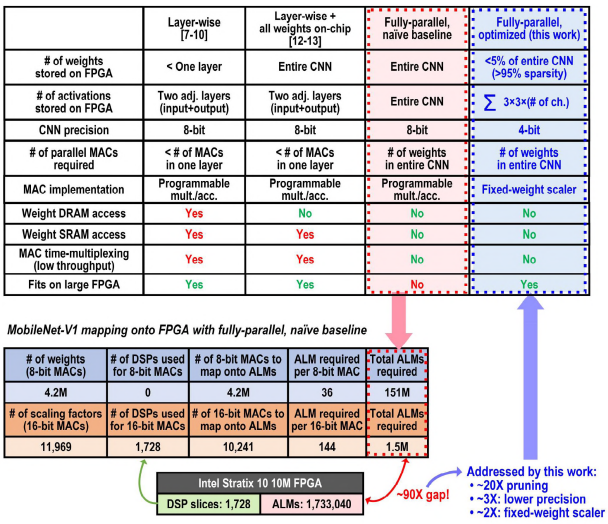

Fig. 1. (Top) Categorization of DNN accelerators on FPGAs. (Bottom) Mapping the entire MobileNet-V1 CNN onto FPGA requires a number of techniques employed collectively in this work.

Fig. 1 (top) shows the categorization of different FPGA-based CNN accelerator schemes. Most of the conventional FPGA-based CNN accelerators [6]–[10] in the literature use off-chip DRAM to store the weights, and the FPGA accelerator performs computation for a single-layer (or a subset of a single-layer) in a time-multiplexed manner. However, the throughput of such designs is often limited by the DRAM bandwidth and the number of multipliers constructed by DSPs. Furthermore, frequently accessing the off-chip memory also introduces high energy consumption [11].

To eliminate DRAM access for DNN inference using a single FPGA, the entire DNN model including weights and activations must be mapped onto the on-chip memory on the FPGA. One of the most well-known compact DNN models

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00010

for the ImageNet dataset is MobileNet [3], which achieves a similar accuracy compared to the conventional VGG-16 [1] (138M weights) or ResNet-18 [2] (11M weights) architectures with significantly less parameters (4.2M weights) and MAC operations. A few prior FPGA designs have fully mapped the compact MobileNet-V1 CNN to a single FPGA without DRAM access [12], [13]. This is possible because recent large-scale FPGAs such as Intel Stratix-10 GX2800 or 10M [14] integrates up to >200Mb of on-chip memory (M20K), which can comfortably hold all MobileNet-V1 weights (4.2M) either in 8-bit or 16-bit precision. Both works [12], [13] store MobileNet-V1 weights in on-chip M20K memory, and load the weights into time-multiplexed multiply-and-accumulate (MAC) units to perform layer-by-layer inference in a pipelined manner.

To fully map MobileNet-V1 onto existing FPGAs and maximize throughput, one MAC unit per weight (i.e. 4.2M MAC units) will be required. Typically DSP blocks are employed for parallel MAC computation, but only thousands of DSP slices exist in large FPGAs (e.g. 1,728 in Intel Stratix-10 10M), and all of these could be used up for the high-precision channel-wise scaling factor computation (Section III-B), which is necessary in 8-bit or lower precision CNNs for better gradient estimation and lower quantization error. This means that all 4.2M MAC units need to be implemented with Adaptive Logic Modules (ALM). Since the mapping of one 8-bit MAC needs 36 ALMs, a total of 151M ALMs are needed for the fully-parallel baseline, which represents a $\sim$90$\times$ gap with the FPGA that has a large number of ALMs (1.73M), as shown in Fig. 1 (bottom).

To bridge this gap, lower precision quantization or pruning can be performed, but previous work [12], [13] did not consider pruning and only lowered the activation/weight precision down to 8-bit. For compact models such as MobileNet, it has been difficult to quantize the activation/weight precision below 8-bit without considerable accuracy loss. A recent algorithm work [15] presented new quantization techniques that lower the precision of MobileNets to 4-bit with minimal accuracy degradation, but did not integrate pruning.

With respect to pruning, element-wise pruning achieves higher sparsity, but the irregular memory access and the index storage overheads, especially for low-precision DNNs, have hindered efficient hardware implementation [16], [17]. Structured pruning schemes [18]–[20] generate sparsity in a hardware-friendly manner by removing a group of parameters in row-/column-wise, block-wise, filter-wise, or channel-wise manner. This leads to efficient hardware acceleration, but the amount of sparsity in structured pruning schemes is typically much lower than element-wise pruning schemes [21].

On the other hand, FixyNN [22] proposed a fixed-weight feature extractor (FFE) design, where the weights are hard-coded in the datapath logic and do not need to be stored in memory. LogicNets [23] also proposed a similar technique to implement neural networks with look-up tables in FPGAs, but the hardware design is only benchmarked for small neural networks with unconventional datasets for jet substructure classification and network intrusion detection.

While FixyNN [22] only employed an FFE for the early layers of CNNs for an ASIC design, in this work, we employ such fixed-weight scalers for the entire CNN layers for an FPGA design. By mapping hard-coded weights in the ALMs of the FPGA, we perform CNN inference of all layers in a fully-parallel, fully-pipelined manner. Contrary to the notion that element-wise sparsity is inefficient for hardware design, one important advantage of the fixed-weight FPGA design (FixyFPGA) is that, element-wise pruning of DNNs can be seamlessly integrated with FixyFPGA design with very high efficiency. This is because pruning out weight elements is equivalent to removing the corresponding hardware operands without introducing any index overhead. This enables us to exploit the high amount of sparsity achievable by the element-wise pruning algorithms [17].

Overall, the main contributions of this work are:

- We present FixyFPGA, a fully-parallel, fully-pipelined, and pruning-friendly FPGA-based CNN accelerator design based on fixed hard-coded weights.
- We optimize the model using aggressive low-precision quantization (e.g. 4-bit) and a high degree of element-wise pruning (e.g. >95%) to achieve heavily-compressed compact DNN models (e.g. MobileNet) for edge computing with limited hardware resources.
- We investigate implementing a number of DNN models with different widths and compression ratios with the fixed-weight scheme onto a single Intel Stratix-10 FPGA chip without any DRAM access.
- We analyze the algorithm and hardware results of DNNs for both image classification tasks (ImageNet, TinyImageNet, and CIFAR-10 datasets) and object detection tasks (Pascal VOC dataset).

The remainder of this paper is organized as follows. Section II describes the fixed-weight FPGA accelerator (FixyFPGA) design. Section III presents the proposed 4-bit MobileNet quantization and high sparsity schemes. In Section IV, we present the experiment results of the FixyFPGA for various CNNs and datasets. The paper is concluded in Section V.

II. Fixed Weight Accelerator Design

FixyFPGA implements CNN models in a layer-parallel fashion, where every non-zero parameter is encoded in the hardware design as a fixed-weight multiplier (scaler). This layer-parallel approach leads to very large improvements in latency and energy, by 1) removing the energy and BW limitations of DRAM, and 2) increasing the number of MACs that can be implemented on an FPGA by $\sim$1.7$\times$ using fixed-weight scalers.

A. Fixed-Weight CNN Datapath

Fixed-weight scalers are significantly smaller, faster and lower energy than full multipliers. Fixed scalers are implemented with a series of hardwired shifts, which are essentially free in hardware, and an adder. The hardware cost is essentially a function of the input operand (activation) bitwidth, and the

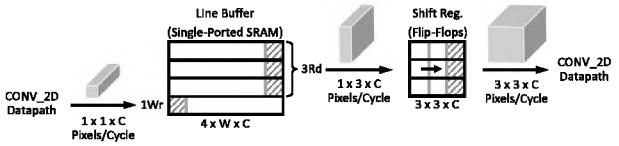

Fig. 2. Hardware implementation of layer-parallel fixed-weight convolution.

Hamming weight (i.e. the number of non-zero bits) of the multiplier (weight), which determines the number of partial products. The adder tree needed to process the flattened output feature map is highly pipelined to achieve high clock frequency. The fixed-weight datapaths are implemented in RTL by embedding the weights into the Verilog as literals. The synthesis tool then generates highly optimized deep sum-of-product datapaths using techniques such as Booth encoding and carry-save addition [24]. Zero weights are simply ignored and do not generate any hardware.

Based on our actual implementation of 4-bit MobileNet-V1 on Intel Stratix 10 FPGA (Section IV), the FixyFPGA scheme consumes 5.87 ALMs per 4-bit scaler on average (i.e. total ALM usage divided by the number of non-zero weights). In comparison, by mapping a 4-bit MAC unit with real multipliers and accumulaters onto the same FPGA, we found that one single non-fixed-weight 4-bit MAC consumes 10.0 ALMs. Therefore, this shows that the fixed design can at least achieve 1.7× reduction in ALMs for each MAC implementation.

B. Fully-Pipelined Activation Buffering

Implementing direct convolution in a layer-parallel configuration requires buffering of activation data flowing through the datapaths. A typical 3×3×C convolution, where C is the number of channels, consumes a 3×3×C input pixel tensor per cycle, but generates only a single small 1×1×C output tensor. Hence, we must buffer several 1×1×C outputs into a larger 3×3×C input for the next layer. Fig. 2 shows how this is implemented using a *line buffer* to store activations at each layer row by row until the required tensor size has been buffered up. Due to the mismatch in input and output tensor dimensions, we need to buffer three full rows before we can start generating the larger output tensors for the following layer. The line buffer itself is implemented on FPGA using on-chip M20K memory and ALMs, and therefore actually requires four independent SRAM banks, so that we can write a single-row patch to one bank per cycle, and read the three-row patch from three banks per cycle, concurrently. After reading/writing the last pixel in a row, the four banks are rotated to overwrite the data associated with the oldest row. Following the SRAM line buffer, a shift-register shifts the convolution window over the feature map, without re-reading data. The shift-register consumes 1×3×C pixels per cycle from the SRAM line buffer and outputs a 3×3×C volume.

C. Deep Freeze Tool Flow

We implemented a tool called *Deep Freeze* [25] to automatically generate a fixed-weight CNN accelerator in Verilog RTL directly from a simple model description in a high-level API. A direct code generation step reads the integer model weights and emits Verilog HDL logic with the weights embedded as immediate constants. Zero weights are skipped entirely. The precision for the intermediate activations is specified as a hardware parameter, along with the accumulator width. The final Verilog is constructed by connecting consecutive combinational datapath stages with buffer stages, which are instantiated from a parameterized Verilog template. The generated Verilog can be directly read in by any synthesis tool for ASIC or FPGA implementation. The generated code is optimized for size and helps reduce compile time, which can be long for such a dense datapath dominated design. During the datapath generation, the bit-widths of the fixed scalers are optimized individually. Deep Freeze also generates a validation suite with testbench for simulation.

III. Aggressive CNN Model Compression

A. Low-Precision Quantization

As we discussed in Section I, 8-bit models consume a large amount of resources and can exceed the capability of many FPGA devices. Therefore, in this work we aim to fully quantize the model to 4-bit for less memory consumption and better resource utilization. Most of the 4-bit quantization schemes focus on the conventional large models such as VGG or ResNet, while we explicitly target compact models such as MobileNet, due to the FPGA hardware constraints. We adopted the progressive quantization method (PROFIT) [15] as the quantization algorithm for MobileNet. By minimizing the output activation instability caused by the quantization level shifting, we quantized the MobileNet-V1 to 4-bit without noticeable accuracy degradation. Such low-precision quantization requires layer-wise adaptive parameters for accurate gradient estimation and quantization error minimization. Generally, the quantization can be formulated into the following steps:

$$W_c = \min(\max(W, -a), a) \quad \text{Clipping} \tag{1}$$

$$S = \frac{2^{n-1}-1}{a} \quad \text{Scaling} \tag{2}$$

$$W_Q = \text{round}(W_c \times S) \quad \text{Quantization} \tag{3}$$

$$W_{QF} = \frac{W_Q}{S} \times \frac{c}{a} \quad \text{De-quantization} \tag{4}$$

Given the input activation X^{l-1} and weight W^l of layer l, The learnable high-precision parameters a and c are involved in both weight and activation quantization. Such floating-point parameters can be multiplied altogether after the convolution operation [26]:

$$Y_Q^l = M \times X_Q^{l-1} * W_Q^l \tag{5}$$

$$\text{where } M = \frac{c_w^l c_x^{l-1}}{a_x^{l+1}} \frac{2^n - 1}{(2^{n-1}-1)(2^n-1)} \tag{6}$$

The only non-integer value is the channel-wise scaling factor M. Depending on the hardware capability, it can be expressed as a high-precision fixed-point coefficient.

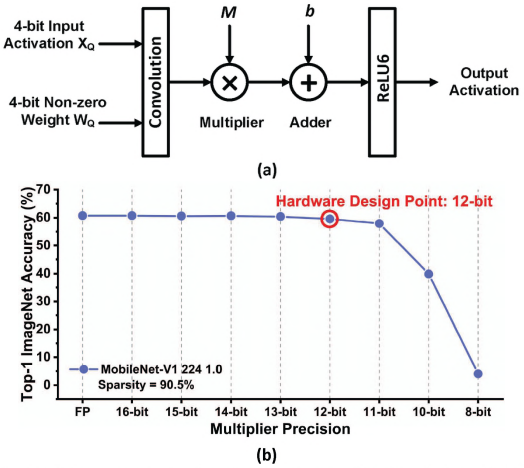

Fig. 3. (a) Inference graph after the batch normalization fusion. (b) Scaling factor precision vs. accuracy for sparse 4-bit MobileNet-V1 1.0 model.

B. Batch Normalization Fusion

The integer-only inference shown in Eq. (5) assumes the batch normalization (BN) parameters are "fused" into the weights before the quantization:

$$W_{fused} = \frac{\gamma W}{\sqrt{\sigma^2 + \epsilon}} \tag{7}$$

$$W_Q = \text{Quantize}(W_{fused}, a_w, c_w) \tag{8}$$

In Eq. (7), γ is the BN weight and σ^2 is the running variance across the batch. Quantizing the fused weights along the output channel dimension can maintain the overall accuracy with high precision quantization [26]. However, it has been shown that such conventional fusion-quantization scheme is difficult to converge with low-precision quantization [15]. Instead of folding BN as part of the weights, we consider it as an adaptive high-precision scaling and shifting process, which can be merged into M. Similarly, the BN bias β will be added after the convolution, as follows.

$$M = \frac{c_w^l c_x^{l-1}}{a_x^{l+1}} \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \frac{2^n - 1}{(2^{n-1} - 1)(2^n - 1)} \tag{9}$$

$$b = \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{10}$$

Merging the BN scaling into high-precision multiplication enables us to avoid the complex BN hardware modules without hurting the pre-trained CNN model accuracy. Fig. 3 shows the inference graph of the trained low-precision sparse model. The scaling factor M and bias b can be computed offline before the FPGA hardware deployment. Such channel-wise parallel with MACs with M and bias b are expansive for FPGA implementation but it is required for accurate low-precision quantization. Given the limited amount of FPGA resources, we swept the precision of the high-precision scaling factors and the results are shown in Fig. 3(b). Based on this, we quantized the high-precision scaling and bias factors to 12-bit fixed-point precision. To recover any accuracy loss, the fused model is fine-tuned with the selected multiplier precision.

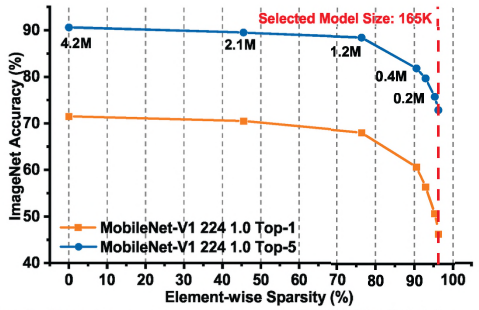

Fig. 4. Sparsity vs. accuracy for 4-bit MobileNet-V1 1.0 model.

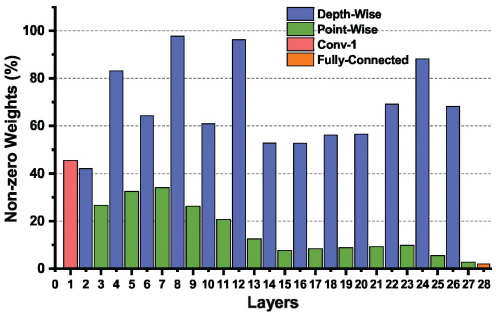

Fig. 5. Percentage of non-zero weights in each layer of 4-bit MobileNet-V1 0.75 after element-wise pruning (total number of non-zero weights is 161K).

C. High Sparsity Element-wise Pruning

FixyFPGA can naturally support element-wise pruning without having random-sparsity index, which is an important advantage that prior pruning works did not exhibit. To achieve high element-wise sparsity, we evaluate the importance of weights by considering the relative magnitude among all the surviving weights in the same layer [17]. We first sort the magnitude of the weights in the ascending order in layer l, and the relative importance of each weight is measured by:

$$\text{score}_{i,j} = \frac{|W_{i,j}^l|^2}{\sum\{|W^l|^2 : |W^l| \geq |W_{i,j}^l|\}} \tag{11}$$

Based on the importance scores of the weights, we globally prune the weights with the smallest scores and gradually increase sparsity, until the targeted sparsity is met.

IV. Experiment Results

A. Experiment Setup

All algorithm experiments are completed with Pytorch API, and the FPGA accelerator generated by the compiler was synthesized using Intel Quartus 20.3. We used Stratix 10 GX 10M FPGA as the target FPGA device, which includes 132 Mb of M20K memory, 1,728 DSP blocks, and 1.73M ALMs. Given the pre-trained 4-bit sparse PyTorch-trained model, we first extract the fixed-point parameters and then generate the corresponding RTL files with the Deep Freeze tool.

B. Algorithm Results

We benchmarked two model architectures on four different datasets. We evaluated the compressed MobileNet-V1 for

TABLE I
EVALUATION OF CNN ACCELERATORS ON STRATIX 10 10M FPGA WITH VARIOUS CNN MODELS AND DATASETS.

Models	# of Params	Top-5 Acc. (%)	Input Size	DSP	ALM	M20K	Freq. (MHz)	FPS	TOPS	Power (W)
MobileNet-V1 1.0 Width [1]	165K	73.32	224 × 224 [2]	1.73K (100%)	1335.9K (77%)	1.39K (21%)	132.85	2.65K	3.01	30.36
		71.59	64 × 64 [2]	1.73K (100%)	1015.4K (59%)	1.32K (20%)	163.11	39.8K	3.74	27.30
MobileNet-V1 0.75 Width [1]	161K	72.87	224 × 224 [2]	1.73K (100%)	1099.1K (63%)	1.11K (17%)	172.92	3.45K	2.27	27.43
		68.41	64 × 64 [2]	1.73K (100%)	1024.6K (59%)	1.11K (17%)	177.43	43.3K	2.32	26.62
MobileNet-V1 0.5 Width [1]	161K	71.47	224 × 224 [2]	1.73K (100%)	824.43K (48%)	0.75K (12%)	163.91	3.27K	1.24	26.90
		68.26	64 × 64 [2]	1.73K (100%)	802.51K (46%)	0.75K (12%)	169.41	41.4K	1.02	26.10
VGG7-C	198K	99.58 (CIFAR-10)	32 × 32	0.36K (21%)	814.98K (47%)	0.29K (4%)	137.29	134.07K	90.95	22.03

[1]Widths of 1.0/0.75/0.5 represent that the number of channels in MobileNet-V1 models are scaled accordingly.
[2]Input image size of 224×224 is for ImageNet dataset, and 64×64 is for TinyImageNet dataset.

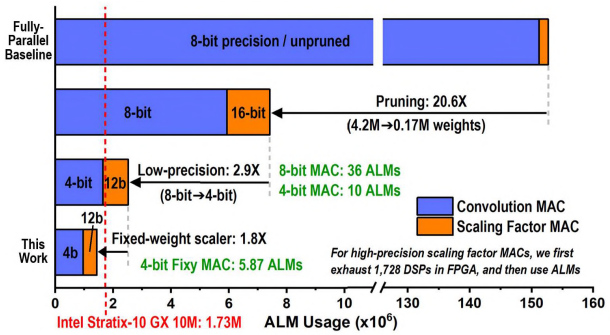

Fig. 6. For fully-parallel implementation of MobileNet-V1 on FPGA, ALM usage is reduced by 107× collectively by pruning, low-precision quantization, and fixed-weight scalers.

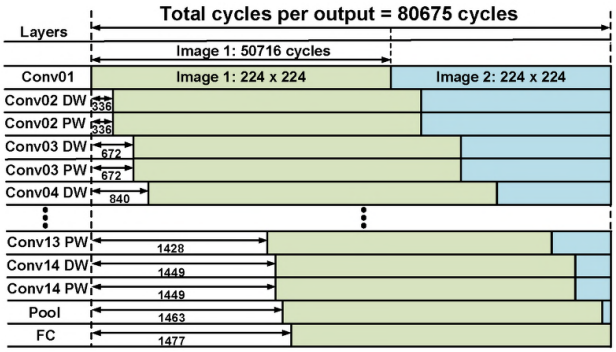

Fig. 7. Layer-wise timing analysis of MobileNet-V1 generated by RTL simulation with 224×224×3 input image for ImageNet.

ImageNet-224, TinyImageNet-64, and Pascal VOC datasets for image classification and object detection tasks. We constructed a customized VGG model VGG-C with the structure of 64C3-P-128C3-P-256C3-P-512C3-P-512C3-FC for CIFAR-10 dataset. The customized 4-bit VGG-C achieved 90.11% accuracy with 198K non-zero weights. To fully map these models onto the target FPGA without any DRAM access, we need to quantize the models down to 4-bit while removing >90% of the weights. It is difficult to simultaneously achieve such aggressive low-precision quantization and high sparsity with one-time training. Therefore, we first performed the aggressive pruning with the full precision model and then progressively quantized the pruned model down to 4-bit. Using the proposed sparsity scheme, Fig. 4 shows the sparsity and accuracy trade-off of the 4-bit MobileNet-V1 1.0 model.

Fig. 5 shows the percentage of the non-zero weights of each layer of MobileNet-V1 0.75 model after pruning. Notably there exists a large sparsity difference between the depth-wise layers and the point-wise layers. According to Eq. (11), such difference indicates that the weights of depth-wise layers have overall higher importance compared to the weights of point-wise layers.

C. FPGA Implementation Results and Analysis

1) Image classification: Fig. 6 shows the how the ∼90× gap pointed out in Section I is addressed in this work, by a series of techniques including pruning with high sparsity (∼20×), 4-bit quantization, and the usage of fixed-point scalers for the entire MobileNet-V1 model. After applying all techniques, the total ALM usage of the proposed FixyFPGA design for MobileNet-V1 model falls under the 1.73M available ALMs in the target FPGA device. The elimination of the DRAM communication also leads to large savings in energy and latency.

Fig. 7 shows the timing diagram for the overall MobileNet-V1 inference, based on RTL simulation. The fully-pipelined activation buffering maximized the computation efficiency by sending the basic $3 \times 3 \times C$ volume to the next layer rather than waiting for the previous computation to complete.

Table I summarizes the resource utilization, throughput, operating frequency, and power consumption with the various CNN models that are trained for different datasets. Every layer of all implemented CNNs are quantized down to 4-bit precision and fully hard-coded into the data logic on FPGA, without any DRAM access. Using the Intel Early Power Estimator, we obtained the power consumption at the junction temperature of 75°C. With the fully-pipelined and fully-parallel FixyFPGA scheme, the latency per image was computed as: $T = \frac{X_H \times X_W}{f}$, where X_H and X_W represents the height and width of the input image. Given the different input sizes of various datasets, the proposed FixyFPGA achieves 3.01, 3.74, and 90.95 GOPS for ImageNet, TinyImageNet, and CIFAR-10 classifications, respectively.

TABLE II
PASCAL VOC OBJECT DETECTION RESULTS WITH DIFFERENT FRONT-END CNN MODELS.

Models	# of Params	mAP (%)	Input Image size	DSP	ALM	M20K	Freq. (MHz)	FPS	TOPS	Power (W)
MobileNet-V1 1.0	152K	54.99	300 × 300	1.73K (100%)	1078.8K (62%)	1.31K (20%)	169.20	1.88K	3.99	28.06
MobileNet-V1 0.75	147K	52.18	300 × 300	1.73K (100%)	993.8K (57%)	1.11K (17%)	196.89	2.19K	2.65	27.41
MobileNet-V1 0.5	148K	51.27	300 × 300	1.73K (100%)	804.5K (42%)	0.75K (12%)	200.76	2.23K	1.24	27.08

TABLE III
COMPARISON TO DIFFERENT FPGA ACCELERATORS FOR MOBILENETS FOR IMAGENET.

Implementations	Model	W / A	Platform	DRAM	Frequency (MHz)	Latency (ms)	GOPS	Frame rate (fps)	Sparsity
DPU [7]	MobileNet-V2	8 / 8	Xilinx Zynq US+	Yes	333	1.23	922	430	Dense
Sparse DLA [27]	MobileNet-V1	FP32	Intel Stratix 10	Yes	257	1.7	0.62	280	71%
HPIPE [13]	MobileNet-V1	16 / 16	Intel Stratix 10	No	430	0.65	-	5157	Dense
TuRF [28]	MobileNet-V1	8 / 8	Intel Stratix V	No	150	4.33	264	231	Dense
TuRF [29]	MobileNet-V1	16 / 16	Intel Stratix V	No	200	0.88	1287	1131	Dense
Tomato [12]	MobileNet-V1	Mixed / 8	Intel Stratix 10	No	156	0.32	3536	3109	Dense
This work	MobileNet-V1	4 / 4	Intel Stratix 10	No	133	0.37	3013	2648	96%

TABLE IV
COMPARISON WITH OTHER OBJECT DETECTION ACCELERATORS BASED ON VOC DATASET WITH BATCH SIZE = 1

Models	W/A	Frame rate	GOPS	DRAM	mAP (%)
Customized MobileNet-SSD [30]	3/4	18 fps	-	Yes	66.40
YOLO-V2 [31]	1/6	60.72 fps	1043	Yes	64.16
Lite-YOLO-V2 [32]	1/1	40.81 fps	-	Yes	67.60
MobileNet-V1 SSD (This work)	4/4	1880 fps	3985	No	54.99

2) Object Detection: We use the compressed MobileNet-V1 CNN model from Table I as the feature extractor for object detection. Table II summarizes the algorithm and hardware results with 300×300 input image size. Similar to prior works [30]–[32], we processed the back-end single-shot detector [33] and non-maximum suppression modules through software simulation and did not implement them in FPGA hardware. The pre-trained 4-bit sparse MobileNet-V1 for ImageNet was fine-tuned by re-training with the Pascal VOC 2007+2012 dataset and tested with VOC 2007. With the focus of high-throughput hardware design, we achieved 3.99 TOPS with the cost of ∼10% mAP degradation (compared with the full precision, unpruned baseline model).

3) Comparison to Prior Works: We compared the hardware performance with previous fully on-chip FPGA-based CNN accelerators with regards to operating frequency, latency, throughput, etc. Unlike the prior works that used on-chip memory to store the weights, our design fully embedded the CNN parameters onto the logic units, which enables us to apply the element-wise pruning without any sparsity index. Therefore, compared with the previous memory-based 8-bit [12], [28] or 16-bit [29] implementations, our design with 4-bit precision and high sparsity will have a large potential for energy-efficiency improvements. Table III shows the comparison results between our design and other recent works. TuRF [29] performed the computation in a layer-by-layer fashion, where the next layer has to wait until the current layer's computation completes. In contrast, with the fully-pipelined and fully-parallel design, our FixyFPGA achieved 3.01 TOPS, which is 2.34× higher than TuRF [29] along with 2.37× latency improvements. Similar to TuRF [29], Tomato [12] stores the power-of-two (POT) weights inside the on-chip memory, streams into the compute engines in a pipelined manner then keeps rolling the output channel to perform the BN multiplications with the given factor. To support such computation, the MAC units should be time-multiplexed. Also, restricting the weights to POT values can lead to considerable accuracy loss in general, due to the rigid resolution of POT quantization [34]. Our proposed design achieved similar hardware performance in a fully-parallel manner, which could be more beneficial to the practical scenarios with high throughput and low-power demands.

Table IV shows the object detection improvements that achieved by our proposed FixyFPGA accelerator. Compared to a prior MobileNet-based VOC object detection accelerator [30], our design achieved over 100× frame rate improvement without using any DRAM. Compared to YOLO-based accelerators [31], [32], our proposed scheme improved the throughput by 3.82×. In addition to the highly-efficient hardware design, our deployed model is also highly sparse, and such high sparsity will improve the power efficiency even further. On the other hand, it is true that such aggressive compression scheme will improve the hardware efficiency with the cost of accuracy degradation. The latest version of Intel Stratix-10 10M FPGA exhibits 2× more ALMs (3.46M) and DSPs (3,456) than the version that we currently used. Using this larger FPGA platform, the proposed FixyFPGA scheme will be able to fit ∼3× more parameters, which will improve the ImageNet inference accuracy by ∼10% (Fig. 3).

V. Conclusion

In this paper, we presented FixyFPGA, a fully-parallel and fully pipelined FPGA-based CNN accelerator design with the objective of compact and high-throughput hardware acceleration. For MobileNet and VGG models for ImageNet, TinyImageNet, CIFAR-10, and Pascal VOC datasets, we performed low-precision quantization down to 4-bit, together with high sparsity of >95%, towards mapping the entire CNN models onto the target FPGA device and eliminating DRAM access. We achieved 3.01 TOPS for ImageNet classification with a low end-to-end latency of 0.37ms, and 3.99 TOPS for Pascal VOC object detection. Compared to prior works, our design achieved 2.34× higher GOPS on ImageNet classification and 3.82× higher frames per second on Pascal VOC object detection.

References

[1] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[4] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *International Conference on Machine Learning (ICML)*, 2019, pp. 6105–6114.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2017, p. 1–12.

[6] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "Automatic Compilation of Diverse CNNs Onto High-Performance FPGA Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 2, pp. 424–437, 2020.

[7] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A High-Performance CNN Processor Based on FPGA for MobileNets," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 136–143.

[8] Y. Yu, T. Zhao, K. Wang, and L. He, "Light-OPU: An FPGA-Based Overlay Processor for Lightweight Convolutional Neural Networks," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020, p. 122–132.

[9] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "HybridDNN: A Framework for High-Performance Hybrid DNN Accelerator Design and Implementation," in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[10] S. K. Venkataramanaiah, H.-S. Suh, S. Yin, E. Nurvitadhi, A. Dasu, Y. Cao, and J.-s. Seo, "Fpga-based low-batch training accelerator for modern cnns featuring high bandwidth memory," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[11] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014, pp. 10–14.

[12] Y. Zhao, X. Gao, X. Guo, J. Liu, E. Wang, R. Mullins, P. Y. K. Cheung, G. Constantinides, and C. Xu, "Automatic Generation of Multi-Precision Multi-Arithmetic CNN Accelerators for FPGAs," in *IEEE International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 45–53.

[13] M. Hall and V. Betz, "HPIPE: Heterogeneous Layer-Pipelined and Sparse-Aware CNN Inference for FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020.

[14] Intel, "Intel Stratix 10 GX/SX Product Table." [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-product-table.pdf

[15] E. Park and S. Yoo, "PROFIT: A Novel Training Method for sub-4-bit MobileNet Models," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 430–446.

[16] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[17] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "A Deeper Look at the Layerwise Sparsity of Magnitude-based Pruning," in *International Conference on Learning Representations (ICLR)*, 2021.

[18] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning Structured Sparsity in Deep Neural Networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[19] G. Srivastava, D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J. Seo, "Joint Optimization of Quantization and Structured Sparsity for Compressed Deep Neural Networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 1393–1397.

[20] L. Yang, Z. He, and D. Fan, "Harmonious Coexistence of Structured Weight Pruning and Ternarization for Deep Neural Networks," *AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 6623–6630, 2020.

[21] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the Granularity of Sparsity in Convolutional Neural Networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.

[22] P. N. Whatmough, C. Zhou, P. Hansen, S. K. Venkataramanaiah, J. Seo, and M. Mattina, "FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning," in *Conference on Machine Learning and Systems (MLSys)*, 2019.

[23] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications," in *IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 291–297.

[24] R. Zimmermann, "Datapath Synthesis for Standard-Cell Design," in *IEEE Symposium on Computer Arithmetic*, 2009, pp. 207–211.

[25] P. Hansen, S. K. Venkataramanaiah, and P. Whatmough, "DeepFreeze: FixyNN Hardware Toolflow," https://github.com/ARM-software/DeepFreeze, 2019.

[26] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.

[27] C. Jiang, D. Ojika, B. Patel, and H. Lam, "Optimized fpga-based deep learning accelerator for sparse cnn using high bandwidth memory," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 157–164.

[28] R. Zhao, X. Niu, and W. Luk, "Automatic Optimising CNN with Depthwise Separable Convolution on FPGA: (Abstact Only)," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 285–285.

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

[30] F. Li, Z. Mo, P. Wang, Z. Liu, J. Zhang, G. Li, Q. Hu, X. He, C. Leng, Y. Zhang, and J. Cheng, "A System-Level Solution for Low-Power

Object Detection," in *IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2019.
[31] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.
[32] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 31–40.
[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 580–585.
[34] Y. Li, X. Dong, and W. Wang, "Additive Powers-of-Two Quantization: A Non-uniform Discretization for Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2020.