Mobile or FPGA? A Comprehensive Evaluation on Energy Efficiency and a Unified Optimization Framework

GENG YUAN* and PEIYAN DONG*, Northeastern University, USA
MENGSHU SUN, Northeastern University, USA
WEI NIU, College of William and Mary, USA
ZHENGANG LI, Northeastern University, USA
YUXUAN CAI, Northeastern University, USA
YANYU LI, Northeastern University, USA
JUN LIU, Carnegie Mellon University, USA
WEIWEN JIANG, University of Notre Dame, USA
XUE LIN, Northeastern University, USA
BIN REN, College of William and Mary, USA
XULONG TANG, University of Pittsburgh, USA
YANZHI WANG, Northeastern University, USA

Efficient deployment of Deep Neural Networks (DNNs) on edge devices (i.e., FPGAs and mobile platforms) is very challenging, especially under a recent witness of the increasing DNN model size and complexity. Model compression strategies, including weight quantization and pruning, are widely recognized as effective approaches to significantly reduce computation and memory intensities, and have been implemented in many DNNs on edge devices. However, most state-of-the-art works focus on ad-hoc optimizations, and there lacks a thorough study to comprehensively reveal the potentials and constraints of different edge devices when considering different compression strategies. In this paper, we qualitatively and quantitatively compare the energy efficiency of FPGA-based and mobile-based DNN executions using mobile GPU and provide a detailed analysis. Based on the observations obtained from the analysis, we propose a unified optimization framework using block-based pruning to reduce the weight storage and accelerate the inference speed on mobile devices and FPGAs, achieving high hardware performance and energy-efficiency gain while maintaining accuracy.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Neural networks**; **Machine learning approaches**.

Additional Key Words and Phrases: DNN model compression, Edge device, Efficient deep learning

*Both authors contributed equally to this research.

Authors' addresses: Geng Yuan, yuan.geng@northeastern.edu; Peiyan Dong, dong.pe@northeastern.edu, Northeastern University, Boston, Massachusetts, USA; Mengshu Sun, Northeastern University, Boston, USA, sun.meng@northeastern.edu; Wei Niu, College of William and Mary, USA, wniu@email.wm.edu; Zhengang Li, Northeastern University, Boston, USA, li.zhen@northeastern.edu; Yuxuan Cai, Northeastern University, Boston, USA, cai.yuxu@northeastern.edu; Yanyu Li, Northeastern University, Boston, USA, li.yanyu@northeastern.edu; Jun Liu, Carnegie Mellon University, USA, junliu2@andrew.cmu.edu; Weiwen Jiang, University of Notre Dame, USA, wjiang2@nd.edu; Xue Lin, Northeastern University, Boston, USA, xue.lin@northeastern.edu; Bin Ren, College of William and Mary, USA, bren@cs.wm.edu; Xulong Tang, University of Pittsburgh, USA, tax6@pitt.edu; Yanzhi Wang, Northeastern University, Boston, USA, yanz.wang@northeastern.edu.

1539-9087/2022/1-ART1 $15.00
https://doi.org/10.1145/3528578

1 INTRODUCTION

The rapid development of DNNs in recent years has led them to become a core enabler for a broad spectrum of application areas, such as computer vision, natural language processing, medical engineering, autonomous driving, and virtual reality. Meanwhile, there is a rising trend showing that the deployment of these applications is shifting from traditional cloud computing platforms (e.g., servers and supercomputers) to edge devices (e.g., mobile and handheld platforms) whose power efficiency is one of the major constraints [1, 21, 32, 37, 48, 61, 79]. Field Programmable Gate Arrays (FPGAs) and mobile devices, as the two most popular substrates in edge devices[1], have established their dominance through the delivery of promising energy/power efficiency and performance. On the one hand, FPGA is known for its high data parallelism, and its data-flow oriented design methodology makes deep and specialized optimizations possible for DNN accelerations. On the other hand, mobile devices are instruction-based computing devices with general-purpose mobile CPU and GPU that provide significant potential for parallel computation.

However, with the ever-increasing demand of DNN accuracy in different application scenarios, there is a significant increase in both the DNN model size and complexity. For instance, both the computation intensity and memory intensity significantly increase when conducting image classification on 3D image segmentation, compared to MNIST, in medical applications (e.g., MRI). Consequently, the computation and memory intensities have become a major obstacle to efficiently deploy DNN applications on edge devices with pleasant energy efficiency and performance.

There exist substantial prior works that target to reduce the computation/memory intensity and improve the execution of DNNs on edge devices through model compression. Based on the high-level techniques they use, one can divide the prior efforts into two bodies: *Quantization-based optimizations* and *Pruning-based optimizations*. In the former [9, 19, 22, 31, 38, 42, 44], high-precision floating-point values are mapped into low-precision values with fewer bits to reduce computation and memory consumption, whereas in the latter [18, 28, 41, 46, 74, 81], the DNN model size is shrank by removing redundant parameters. These two strategies are platform-independent and can potentially work collaboratively in some scenarios. First, FPGAs are good at handling calculations with linear-quantized (e.g., fixed-point) data as well as bit shift calculations with non-linear-quantized (e.g. power-of-2) data. As a result, a large block size is required for DNN implementations on FPGAs for maximum computation parallelism, whereas pruning introduces irregularity into the DNN models and is not compatible with large block sizes. Second, in mobile devices (especially mobile GPU), fixed-point computation is not natively supported. Meanwhile, pruning can achieve a higher compression rate compared with quantization, which is especially important for bandwidth-bound DNNs.

In this paper, we ask several fundamental questions regarding accelerating DNN executions on FPGAs and mobile devices. First, the fundamental designs of FPGAs and mobile devices are different, we want to ask how these two platforms compare with each other considering the optimizations? Second, it is generally acknowledged that FPGAs have better energy-efficiency compared to general computing mobile devices [11, 63]. Is this knowledge holding true for DNN executions as well? Third, it is not clear the maximal potential of applying quantization and pruning on FPGAs and mobile devices. What are the maximum benefits one can obtain from applying these optimizations on both platforms?

To answer these questions, we conduct a thorough analysis of FPGA-based and mobile-based solutions under both baseline executions as well as optimized executions where state-of-the-art optimizations are applied. Based on our analysis, we qualitatively and quantitatively compare FPGAs and mobile devices. We observe that, for DNN executions, it is very difficult for FPGAs to beat mobile devices when considering both energy-efficiency and model accuracy. In fact, even with state-of-the-art quantization applied to FPGAs, the energy-efficiency of FPGAs is still lower compared to mobile devices without pruning being applied. To further explore the benefit

[1]We focus on off-the-shelf computing devices in this paper.

of weight pruning, we propose a unified, hardware friendly pruning strategy that can be effectively applied to both mobile devices and FPGAs, along with compiler-assisted acceleration framework. For both mobile devices and FPGAs, the proposed pruning and acceleration framework can achieve high hardware performance gain while maintaining accuracy. We find that, under a similar accuracy, weight pruning provides higher benefits to mobile devices than FPGAs, which further enlarges the energy efficiency advantage of mobile-based solutions over FPGA-based ones. This paper makes the following contributions:

- We conduct a thorough analysis of modern FPGA-based and mobile GPU-based DNN executions. The analysis is established through a detailed characterization of energy efficiency, performance, and accuracy for both FPGAs and mobile devices.
- We qualitatively and quantitatively compare the FPGA-based and mobile-based DNN executions. Our comparison is conducted in a step fashion where we start with the baseline execution, and then apply optimizations (i.e., quantization and pruning).
- Based on our observation, we conclude that with the technology currently in wide usage, mobile GPU-based DNN executions are more energy-efficient than FPGA-based DNN executions, unless the binary or ternary quantization is adopted on FPGA at the cost of significant accuracy loss.
- We further propose a block-based pruning scheme and a unified optimization framework to reduce the weight storage and accelerate the inference speed.
- Different from the existing pruning schemes, our proposed pruning scheme is friendly to both mobile GPU-based and FPGA-based DNN executions. And the corresponding compiler optimizations are designed.
- The experimental results demonstrate that our proposed framework outperforms the state-of-the-art mobile-based and FPGA-based DNN acceleration frameworks.

2 BACKGROUND

2.1 Weight Quantization

DNN quantization has become an effective compression technique for acceleration. It maps high-precision floating-point values into low-precision values. Linear quantization schemes with uniform quantization levels have been the focus of many works, including fixed-point [9, 10, 19, 22, 38, 84], ternary (with values −1, 0, +1) [31, 40, 85], and binary (with values −1, +1) [12, 13, 44, 65]. Additionally, a non-linear logrithmic quantization scheme called power-of-2 (P2) has been utilized on weight parameters in some studies [39, 42, 54, 78, 83]. A variant of the P2 scheme is sum-of-power-of-2 (SP2) with the summation of two P2 numbers as a weight value for more precise data representation [7].

On FPGAs, weight quantization is a natural fit. Besides storage reduction, the additional benefits include (1) the DSP on FPGA can support multiple multiply-and-accumulate (MAC) computations with appropriate weight (and activation) quantization, and (2) the look-up table (LUT) computing resources can support low-precision computing [24, 25, 35, 36, 47, 48, 55, 58, 71]. Low-bit-width fixed-point quantization is achieved in [24] through greedy solution to determine the radix position of each layer for quantization, and in [71] with a hybrid quantization scheme that allows different bit-widths for weights to provide more flexibility. Binarized Neural Networks (BNNs) can be implemented with XNOR gates to execute multiplications [25, 55, 58], and replacing zero padding with odd-even padding enables it to realize a fully binarized neural network accelerator [25]. The implementation with P2 quantization has been explored in [48].

On mobile devices, the quantization technique is not commonly adopted in mobile-based DNN inference accelerations mainly because of two reasons: (1) Mobile GPUs do not natively support the quantization technique and hence the low-bit or fixed-point computations [2]; (2) Even though the mobile CPU can support as low as 8-bit quantization, but it is not a desired computing device for DNN accelerations. This is because: first, the higher power of mobile CPU (3W) compared to mobile GPU (1.5W) makes it less energy efficient, and second,

the mobile CPU is usually occupied by the operating system and other background programs. Thus, quantization is less favored by mobile-based DNN accelerations compared with FPGAs.

2.2 Weight Pruning

Weight pruning for DNNs has been widely studied these years, eliminating redundancies of the DNN model to reduce both the storage and computation consumption in DNN inference. The existing weight pruning researches can be categorized into two major groups according to the pruning scheme: Unstructured pruning [14, 26, 27, 53] and structured pruning [18, 28, 30, 41, 46, 49, 72, 74, 81].

Unstructured pruning, also called irregular pruning, removes weights at arbitrary location as shown in Figure 1 (a). The early work [27] focused on the pruning of fully connected (FC) layers by using a heuristic approach to prune the weights with the least magnitudes. The later work [26] (dynamic network surgery) extended to the convolutional (CONV) layers which are the most compute-intensive. Extensions of iterative weight pruning, such as [14] (NeST) and [53], use more delicate algorithms such as selective weight growing and pruning to achieve a higher compression rate. Though unstructured pruning can significantly decrease the weights number in DNN model, the resulted sparse matrix requires extra index storage and degrades the parallel implementations, which results in limited improvement of hardware performance.

To overcome the limitation in unstructured, irregular weight pruning, much work [18, 28, 30, 41, 45, 46, 49, 72, 74, 81] studied the structured pruning at the level of filters and channels as shown in Figure 1 (b). The pruned model would maintain the network structure which can be supported by the prevalent DNN acceleration framework such as TVM and TensorFlow-Lite. Early work [30, 72] incorporate ℓ_1 or ℓ_2 regularization in loss function to solve filter/channel pruning problems. The method in [28] selects filters based on ℓ_2 norm and updates the filters that have been previously pruned. [41, 81] incorporate the advanced optimization solution framework ADMM (Alternating Direction Methods of Multipliers) to achieve dynamic regularization penalty, thereby improving accuracy. [29] proposes to adopt Geometric Median, a classic robust estimator of centrality for data in Euclidean spaces. [45] incorporates simulated annealing to automatically search the hyperparameters in weight pruning and accelerates the inference implementation as a further step.

On FPGAs, acceleration with pruning techniques have been investigated in several studies [21, 32, 34, 61, 79]. Generally, FPGA implementations of DNNs adopt a large tiling size in the filter and channel dimensions for high computation parallelism, but the irregularity in DNNs introduced by pruning will severely degrade the hardware utilization and prevent the FPGA implementations from achieving the maximum parallelism. Coarse-grained pruning like filter pruning employed in [21] is more hardware-friendly than unstructured pruning adopted in [79], but will result in relatively high accuracy loss.

On mobile devices, state-of-the-art mobile acceleration frameworks, including TF-Lite [1], TVM [8], and Alibaba Mobile Neural Network (MNN) [37], have limited support of weight sparsity of DNNs. They only support filter/channel pruning/sparsity in nature, while does not accommodate the other versatile sparsity schemes. In fact, mobile CPU and GPU are instruction-based, general-purpose computing devices, with flexible compiler-level software code generation. This provides a natural advantage of supporting weight pruning techniques through compiler-assisted software optimizations. Recently, pattern-based pruning is proposed to prune the weights at an intra-kernel level by enforcing the locations of the remaining weights in a 3×3 convolutional kernel to form a specific kernel pattern [50, 60]. In pattern-based pruning, each kernel pattern reserves 4 non-zero weights out of the original 3×3 kernel to match the single-instruction multiple-data (SIMD) architecture of embedded CPU/GPU processors to maximize the hardware throughput.

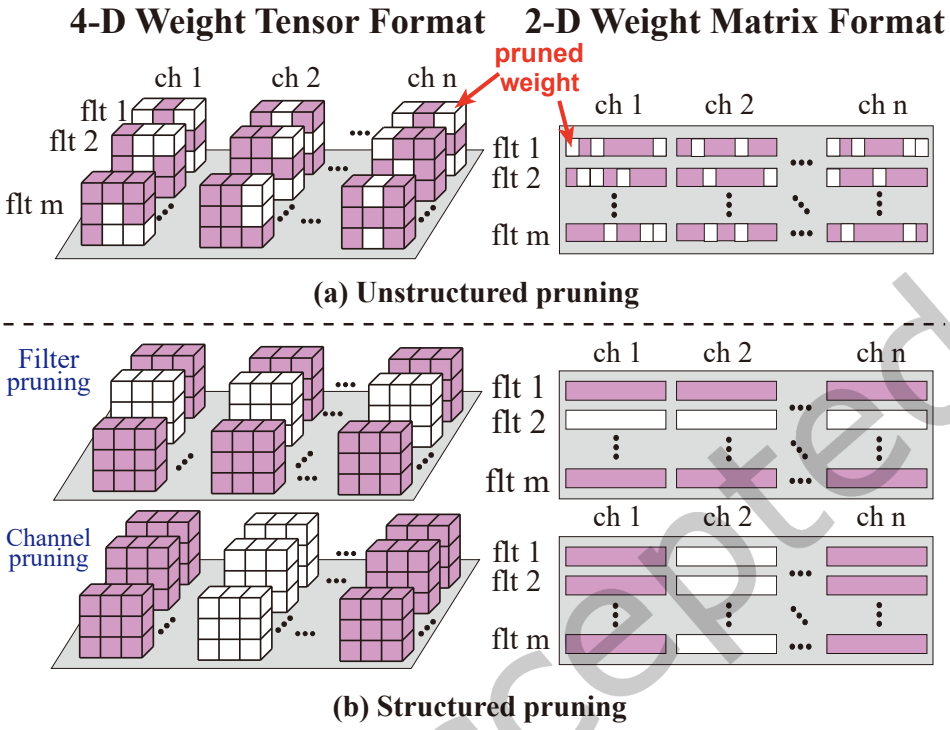

Fig. 1. Different types of weight pruning in 4D tensor and 2D matrix formats.

3 COMPARISONS OF FPGA-BASED AND MOBILE-BASED SOLUTIONS

3.1 Methodology

To make a thorough analysis of modern FPGA-based and mobile-based DNN executions, we make a qualitative and quantitative comparison based on the state-of-the-art FPGA-based and mobile-based DNN solutions in terms of energy efficiency and accuracy.

As shown in Table 1, we collect recent year's representative FPGA-based DNN acceleration solutions for the image classification task on ImageNet dataset [15], and the object detection task on Pascal VOC dataset [20], respectively. We also summarize the bit-width, frequency, number of DSPs, FPGA chip power (which is a better indicator than the whole board power), and FPGA platform used in each collected solution.

For the mobile-based solutions, we select three modern DNN acceleration frameworks including MNN [37], TVM [8], and TF-Lite [1]. A Samsung Galaxy S20 smartphone is used as our mobile device, which integrates a Qualcomm Adreno 650 mobile GPU in the Qualcomm Snapdragon 865 processor using 7nm technology. Since its mobile GPU has lower power (1.5W) than the mobile CPU (3W), and to avoid the impact of resource contention caused by other programs running on the CPU during the measurement process, all the mobile-based results are tested using mobile GPU operating using 16-bit floating-point representations. Note that some high-end mobile devices integrate an NPU, which generally leads to higher energy efficiency. In this paper, we mainly focus on DNN executions using mobile GPU.

Table 1. **Representative FPGA DNN acceleration solutions for image classification tasks on ImageNet dataset and object detection tasks on Pascal VOC dataset.** A single bit-width value indicates the bit-width for both weight and activation; w[b_1]a[b_2] means b_1-bit fixed-point weight quantization and b_2-bit fixed-point activation quantization; P2 stands for power-of-2 weight quantization; SP2 stands for sum-of-power-of-2 weight quantization; mixed (·) indicates combining multiple weight quantization schemes in a single layer.

Implementation	Bit-width	Frequency (MHz)	DSP (Used/Total)	FPGA Chip Power (W)	Platform
ImageNet					
[66]	32	150	140/220	2	XC7Z020
[64]	16	150	780/900	4	XC7Z045
[77]	16	150	2833/3632	26	Virtex 690t
[76]	32	200	224/256	13.18	Stratix-V
[23]	8	214	198/220	3	XC7Z020
[80]	16	385	2756/3036	37.5	Arria-10 GX1150
[52]	w8a16	150	1518/1518	21.2	Arria-10 GX1150
[69]	w8a4	150	1452/2520	8	ZCU102
[4]	8	150	1278/1687	15.6	Arria-10 SX SoC
[70]	16	125	198/220	1.75	XC7Z020
	16	125	855/900	4	XC7Z045
[75]	8	200	704/840	8.5	XCK325T
[7]	1) P2; 2) SP2;	100	20/220	2.5	XC7Z020
	3) mixed (4/SP2) 4) mixed (4/8/SP2)		220/220		
Pascal VOC (2007 + 2012)					
[17]	8	200	3600/3600	23	ADM-PCIE-7V3
	mixed (8/P2)	200	3600/3600	21	ADM-PCIE-7V3
[62]	w1a3	NA	360/360	6	ZU3EG
[57]	1	299.97	377/2520	4.5	ZCU102
[56]	w-ternary, a1	150	114/360	1.2	ZU3EG
[82]	32	200	800/NA	1.17	XC7Z045
[73]	8	NA	1024	13	KU115
[59]	w1a6	200	168/2800	8.7	VC707
	w1a6	200	272/2800	18.29	VC707

The VGG16, ResNet18, MobileNet networks, and YOLO networks are used for the comparison. We mainly conduct comparisons of model accuracy and execution energy efficiency. For the former, we use the top-1 accuracy on the ImageNet dataset and mean Average Precision (mAP) on the Pascal VOC dataset, whereas for the latter, we use Giga-operations per watt (GOPS/W) as the metric for the energy efficiency.

3.2 Baseline Comparisons

First, in order to investigate the baseline performance (energy efficiency) of FPGA and mobile devices, we select representative solutions from both FPGA and mobile sides. Note that, we exclude the use of pruning techniques in this baseline comparison (hence the original, unpruned network is executed). 16-bit floating-point computation is utilized for the mobile baseline, as further quantization is not supported [2]. 32-bit or 16-bit fixed-point quantization is selected for FPGA baseline, and further low-bit quantization (i.e., 8-bit quantization or beyond) is considered to be extensions that will be discussed in Section 3.3.

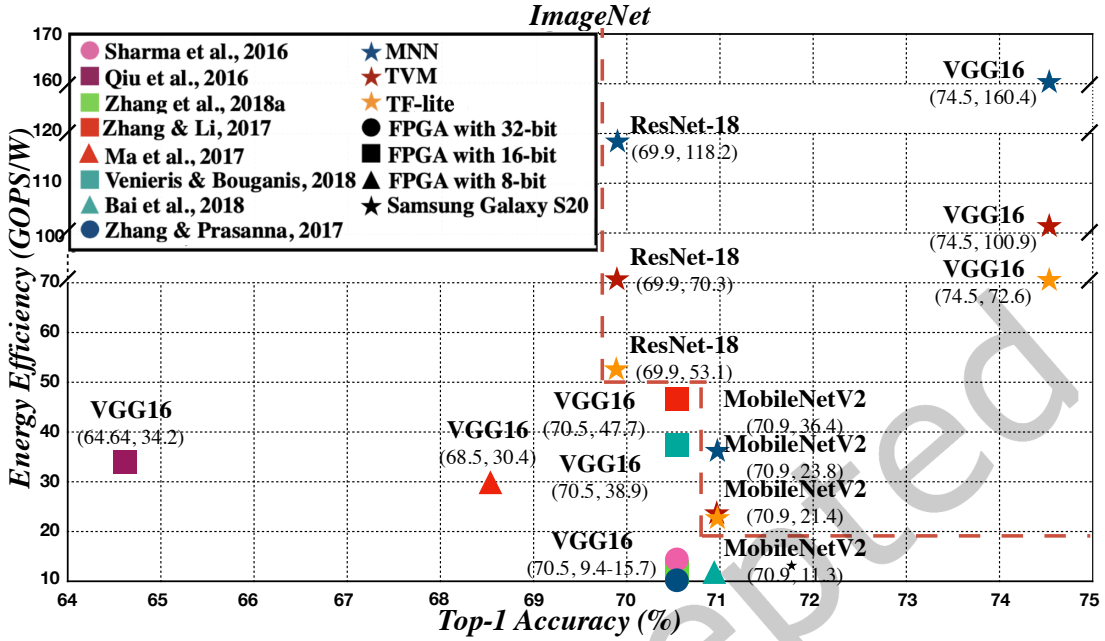

Fig. 2. Energy efficiency (GOPS/W) vs. accuracy on Mobile Phone & FPGA on CNNs with ImageNet dataset. Comparisons are among baseline implementations.

Figure 2 shows the energy efficiency comparison of FPGA-based and mobile-based solutions in terms of GOPS/W, – the star shapes in the figure represent mobile-based solutions, whereas the circle, square, and triangle shapes represent FPGA-based solutions using 32-bit, 16-bit, and 8-bit weight representations, respectively. Since the model accuracy is considerably affected by the training recipe used (e.g., the hyperparameters, the augmentations, the training tricks). Even with the same model structure (e.g., VGG16 in Figure 2), the accuracy can vary a lot. Therefore, we do not focus on the accuracy comparison. Some designs such as Zhang & Li, 2017 [80], Ma et al.,2017 [52], and Bai et al.,2018 [4] use high-end FPGAs (e.g., Virtex 690t and Arria-10 GX1150) that have more resources and computing power (as shown in Table 1), so they may not be considered edge devices. For MobileNetV2, an 8-bit FPGA-based solution is included for comparison since there is not a valid FPGA solution with 16- or 32-bit weight representation. The red dash line in the figure separates the results between the solutions based on FPGAs and mobile devices. From Figure 2, we make the following key observations.

Observation 1. We observe that both the mobile-based and FPGA-based solutions achieve higher energy efficiency on VGG16 than on MobileNetV2. This is because VGG16 has relatively regular network architecture with uniform 3×3 kernel and requires more intensive computations for each layer, which can better leverage hardware parallelism. The MobileNetV2 introduce skip-connections and downsampling layers with 1×1 kernels, which decrease the regularity of the network structure, and lower the energy efficiency in GOPS/W compared to VGG16.

Observation 2. From the mobile device perspective, three mobile-based solutions are compared (i.e., MNN, TVM, and TF-Lite). Since the same models are used, the accuracy is the same for all three solutions. MNN consistently achieves the highest energy efficiencies, outperforming TF-Lite by 2.2× and 1.7× on VGG16 and MobileNetV2, respectively. All these three frameworks can be used on different types of devices. The reason that

MNN outperforms TVM and TF-Lite is that the MNN made more superior optimizations dedicated to mobile devices.

Observation 3. From the FPGA perspective, the [80] is considered the best baseline solution with the highest energy efficiency and accuracy over other solutions on VGG16. This is because that [80] effectively leverages the large number of DSPs in the high-end FPGA, and achieves high operating frequency by efficient implementation. Note that we cannot find a FPGA-based solution on MobileNetV2 without using quantization, [4], which uses the 8-bit quantization, is used for the comparison on MobileNetV2.

Observation 4. Comparing the FPGA-based solutions to mobile-based solutions with the same network model, one can observe that the energy efficiency of mobile-based solutions are 1.5× ∼ 3.4× and 1.9× ∼ 3.2× higher than the best FPGA-based baseline solutions on VGG16 and MobileNetV2, respectively. This contradicts people's general belief, i.e., FPGAs would have better energy-efficiency compared to general computing mobile devices [11, 63]. One reason is that mobile devices always adopt the most advanced technologies. For example, the Samsung Galaxy S20 smartphone adopts the Qualcomm Snapdragon 865 processor using 7nm technology, whereas the high-end FPGAs such as the Arria-10 are still using 20nm technology. This provides mobile devices a natural advantage in terms of energy efficiency and operating frequency.

Takeaway. The network with higher structural regularity and more intensive computation is more likely to have higher energy efficiency in terms of GOPS/W on both FPGAs and mobile devices. While the effective use of DSPs can also improve FPGA execution energy efficiency, without the help of appropriate compression-based optimizations, the FPGAs are not competitive with mobile devices in *all* DNN executions we have studied.

However, we cannot yet make the final conclusion about the potential of FPGAs and mobile devices. From Figure 2 one can also observe that the FPGA solutions with 16-bit weights show a significant advantage over FPGA solutions with 32-bit weights. This indicates a great potential of weight (and activation) quantization in improving the energy efficiency of FPGA-based designs. We will analyze the effect of quantization in details in Section 3.3.

3.3 Optimizations on FPGA Solutions

3.3.1 Effectiveness Analysis of Various Optimizations. In this section, we analyze the performance of various FPGA designs and explore the potential of quantization and other optimizations for improvement in energy efficiency of FPGA designs. Figure 3 compares the energy efficiency (GOPS/W) and accuracy of state-of-the-art FPGA-based solutions for DNNs on ImageNet dataset. The performance of FPGA implementations can be affected by the following factors:

(1) **The type of FPGA device adopted in design.** By comparing the two designs in [70], which adopt identical design methodology but on different types of FPGA boards, it can be seen that the design on the XC7Z045 device achieves higher energy efficiency (38.9 GOPS/W) than the one on the XC7Z020 device (27.7 GOPS/W) given the same operating frequency, as XC7Z045 has more available DSPs to accommodate more computations in each clock cycle while maintaining relatively low power consumption.

(2) **Quantization with different bit-widths.** Lower bit-width quantization generally leads to higher energy efficiency. Take the VGG16 designs shown in Figure 3 for instances, the 32-bit implementations attain energy efficiency from 9.4 to 15.7 GOPS/W, and for 16-bit this range is 13.6 to 47.7 GOPS/W. The design in [23] is built on XC7Z020 with fewer DSPs than other implementations, but the 8-bit quantization assists this design to achieve comparable energy efficiency (41 GOPS/W) with the 16-bit VGG16 designs on larger FPGAs. This could be explained by the DSP consumption for computations in different bit-widths. While one MAC operation for 32-bit floating-point data may utilize five DSPs [33], one MAC for 16-bit fixed-point data only requires one DSP. Quantization with lower bit-width will further enhance the performance by using one DSP for multiple operations, as discussed below.

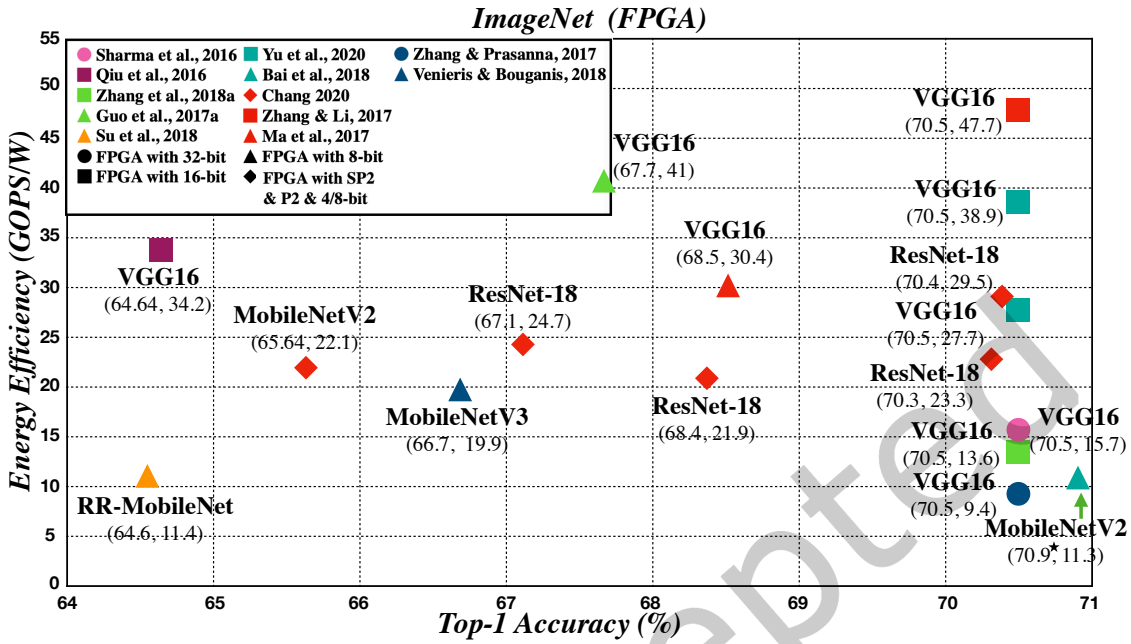

Fig. 3. Energy efficiency (GOPS/W) vs. accuracy on FPGA for DNNs on ImageNet dataset.

(3) **Utilizing one DSP for multiple multiplications.** Low bit-width (eg., 8-bit or even lower bit-witdh) quantization provides oppotunity for the FPGA implementations to manage more than one multiplications in each DSP. For instance, the design in [75] leverages 8-bit quantization and decomposes each Xilinx DSP48E1 25-bit $\times$ 18-bit multiplier into two 8-bit $\times$ 8-bit multipliers. This makes the MobileNetV3 implementation, although with more complicated network architecture, enjoy higher energy efficiency (19.9 GOPS/W) than the MobileNetV2 design in [4] (11.3 GOPS/W).

(4) **Power-of-2 (P2) and Sum-of-power-of-2 (SP2) quantization that utilizes look-up tables (LUTs) only.** The P2 scheme replaces each multiplication with a bit shift operation consuming LUTs which have higher energy efficiency and integration density than DSPs on FPGA. SP2 proposed in [7] provides better data representation by adding two P2 results to replace one multiplication and therefore resulting in higher accuracy than P2. P2 consumes fewer LUTs than SP2, thus attaining higher energy efficiency, but incurs more significant accuracy loss. As a result, P2 and SP2 achieve energy efficiency of 24.7 GOPS/W and 21.9 GOPS/W, respectively, comparable with VGG16 designs.

(5) **Mixed quantization scheme and mixed precision.** The designs mentioned above are based on either fixed-point data only or P2/SP2 scheme only. One shortcoming is that these designs only consume one type of computation resources in FPGAs (i.e., either LUTs or DSPs), and thus cannot fully utilize the resources of FPGAs. In [7], the mixed-scheme (MS) design combines the 4-bit fixed-point scheme and 4-bit SP2 scheme to exploit both DSP and LUT resources on FPGA fully and balancedly. The ratio of fixed-point to SP2 quantization is determined according to the hardware resource analysis for a certain FPGA. The mixed-scheme and mixed-precision (MSP) design further introduces 8-bit fixed-point quantization into MS. Among ResNet-18 implementations, the one based on MSP performs the best (29.5 GOPS/W), better than MS (23.3 GOPS/W), because MSP has higher capability

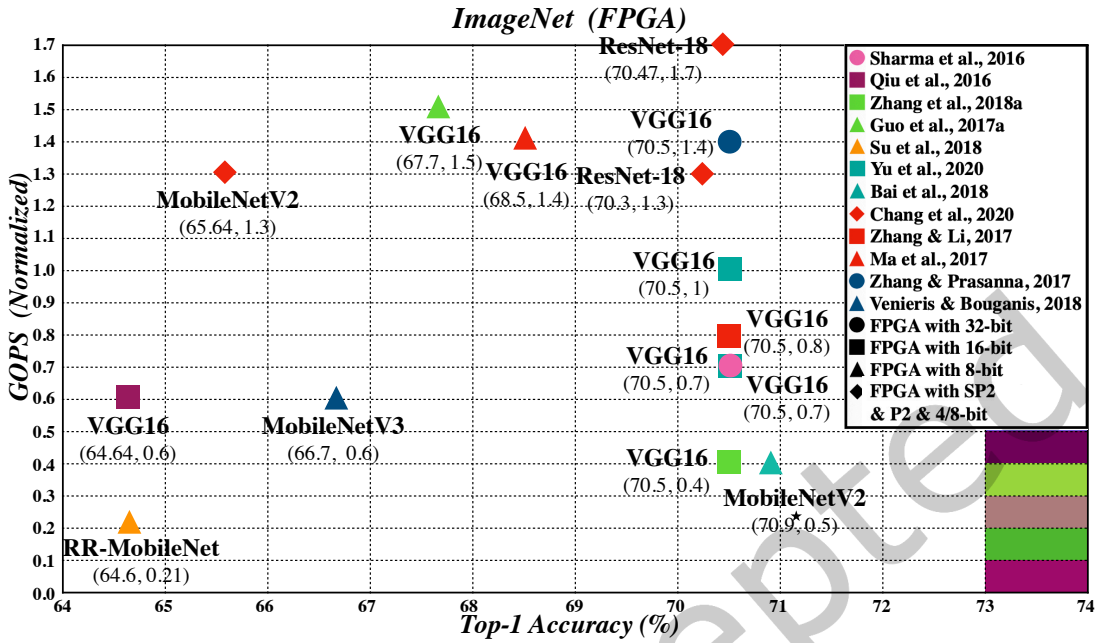

Fig. 4. Normalized GOPS on FPGA for CNNs on ImageNet dataset.

to maintain the accuracy and the first and last layers are quantized, while these layers are not quantized in the MS design. As for the MobileNet-V2 implementation based on MS, the energy efficiency could reach 22.1 GOPS/W.

3.3.2 Analysis of Normalized GOPS. As discussed above, the energy efficiency is affected by the FPGA board adopted as the implementation platform. To eliminate the impact of platforms as much as possible and illustrate the efficacy of quantization with various optimization techniques, we calculate the "nominal" GOPS for each design through multiplying the number of used DSPs by the actual operating frequency, based on the following assumptions: (1) The DSP utilization rate is a key factor for the energy efficiency of the FPGA design; (2) Each DSP manages one fixed-point multiplication in most cases. We then normalize the GOPS in each design so that the normalized GOPS is the ratio of the actual GOPS to the nominal GOPS.

Figure 4 displays the normalized GOPS versus accuracy on FPGA for DNNs on ImageNet dataset. Ideally, if all DSP are utilized, the normalized GOPS should be 1. A normalized GOPS less than 1 might result from two main reasons: (1) DSPs are required to handle other DNN operations than MACs, like the pooling operations that exist in almost all DNN models; (2) The irregularity in some models (especially the MobileNet models) degrades the overall performance. Another interesting observation is that in some cases the normalized GOPS is larger than 1. This can be achieved through optimization techniques like integrating multiple multiplications in one DSP, which is implemented in [7, 52]. Specifically, the work [52] uses an Arria-10 FPGA, where each DSP can support two 18-bit × 18-bit multiplications, and the DSP utilization rate in this design reaches 100%. In addition, the quantization schemes in [7] all achieve normalized GOPS above 1 due to the low-bit-width precision, and the MSP scheme with normalized GOPS of 1.7 performs the best among all designs. Another technique for high normalized GOPS is managing DNN operations in the frequency domain through Fast Fourier Transform (FFT) [76].

From Figures 3 and 4, we conclude that the potential optimization directions include (1) utilizing each DSP for multiple multiplications, and (2) quantizing with mixed scheme and mixed precision. However, the improvement

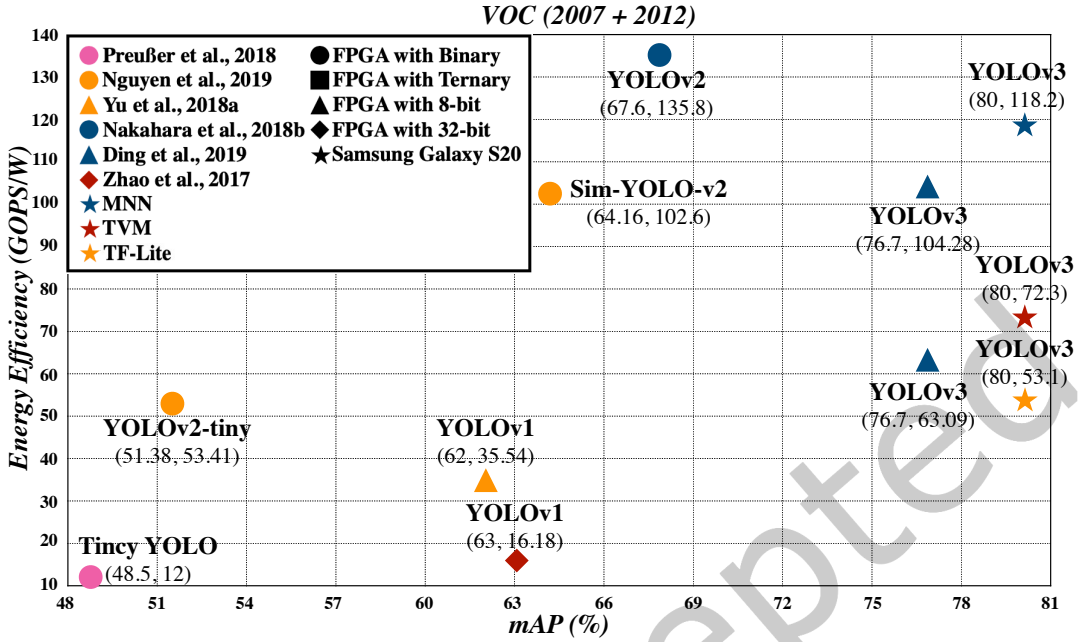

Fig. 5. Energy efficiency (GOPS/W) vs. accuracy on Mobile Phone & FPGA on YOLOs with VOC (2007+2012) dataset.

resulting from these techniques is still not sufficient for FPGA implementations to compete with mobile counterparts. Considering ResNet-18 designs as examples, the best-optimized FPGA design is based on MSP which achieves energy efficiency of 29.5 GOPS/W. However, the best baseline mobile design, namely MNN, achieves 118.2 GOPS/W that outperforms FPGA by 4×.

Here, we raise the following questions: (1) Is this the best result that can be achieved through quantization techniques? (2) Is it possible to make FPGA-based solutions achieve better energy efficiency than mobile-based solutions with the technology currently in use? In the subsequent sections, we quantitatively answer these two questions.

3.3.3 Analysis of Binary & Ternary Quantization. With more aggressive quantization strategy, i.e., **binary or ternary** weight quantization, the original MAC operations between input data (activation) and weights can be performed as addition/subtraction operations, which only need to use LUTs instead of DSPs. Thus, the energy efficiency can be further boosted.

Since very few FPGA-based works adopt binary/ternary quantization on the ImageNet dataset, we compare the energy efficiency of FPGA-based implementations that use binary quantization with non-quantized mobile-based solutions on the Pascal VOC dataset for the object detection task, as shown in Figure 5. Note that we also include other representative works that are not using binary/ternary quantization to make more comprehensive comparisons.

Observation 1. It can be observed in Figure 5 that with binary quantization, [57] successfully outperforms the best mobile-based solution MNN in energy efficiency by 1.15×, and [59] also achieves 102.6 GOPS/W energy efficiency, which is slightly lower than MNN. However, compared to the original mAP of YOLOv2, there is a 9.2% and 12.64% mAP drop in [57] and [59], respectively.

Observation 2. Besides the solutions with the binary weight quantization, the design [17] also adopt several optimizations. It incorporates the block-circulant [16] compression technique and replaces the original multiplication operations with Fast Fourier Transform (FFT)-based fast multiplications to reduce the weight storage and computation complexity. Moreover, higher energy efficiency can be achieved by mixing the 8-bit fixed-point quantization with P2 quantization to explore both DSP and LUT resources, where the other result only adopts 8-bit fixed-point quantization. With mixed quantization scheme, the energy efficiency is improved by 1.65× compared to the 8-bit fixed-point quantization solution. This result more intuitively proves the effectiveness of mixed quantization scheme in improving the FPGA-based design energy efficiency. Although [17] achieves comparable energy efficiency (i.e., 11% lower than MNN) under multiple optimizations, it needs to sacrifice 3.3% model mAP (as reported in [17]).

Takeaway. Based on our observations, we conclude that FPGA-based solutions can achieve higher energy efficiency compared to the mobile-based solutions when adopting binary or ternary quantization while leading to a significant accuracy loss. Otherwise, with the technology currently in wide usage, mobile-based solutions are more energy-efficient than FPGA-based solutions.

4 WEIGHT PRUNING OPTIMIZATION FRAMEWORK AND EVALUATIONS ON MOBILE DEVICES AND FPGAS

As we discussed in Section 3, even with aggressive optimizations, FPGA-based solutions cannot achieve comparable energy efficiency as a mobile-based solution without significant sacrifice of accuracy. We are still wondering how good performance can be achieved for mobile devices with appropriate optimizations.

Since quantization techniques are not favored by mobile GPU as mentioned in Section 2.1, we will adopt weight pruning to evaluate the improvement on mobile devices. And we will also evaluate the benefit of weight pruning on FPGAs. The conventional pruning schemes, including unstructured pruning and coarse-grained structured pruning, are either not hardware friendly or introducing non-negligible accuracy loss, so they are not desired to be used in DNN accelerations. To solve that issue, the pattern-based pruning scheme is proposed in [51, 60], which is tailored to mobile devices. It prunes weights by enforcing the locations of the remaining weights in a 3×3 convolutional kernel to form a specific kernel pattern. And each kernel pattern reserves 4 non-zero weights out of the original 3×3 kernel to match the single-instruction multiple-data (SIMD) architecture of mobile processors to maximize the hardware throughput.

However, the inherent disadvantage of the pattern-based pruning is that it can only be applied to 3×3 CONV layers. In many networks, a large portion of weights and computations are contributed by non-3×3 CONV layers (e.g., 1×1 CONV layers and fully connected layers).

Thus, we propose a DNN acceleration framework, which mainly consists of two parts: 1) a unified *block-based pruning scheme* for both mobile devices and FPGAs; 2) the *compiler-based optimization* for mobile devices. Our proposed block-based pruning scheme that combines the advantage of both unstructured pruning and structured pruning achieves high accuracy and hardware friendly simultaneously. It can be generally applied to all types of DNN layers and is compatible with both mobile devices and FPGAs. With our compiler-based optimization, the computation and storage reduction provided by weight pruning can be fully leveraged by mobile devices, and a more efficient DNN execution code can be generated for faster DNN inference.

4.1 Our Compiler-assisted Acceleration Framework for Mobile Devices

To achieve the best performance for different DNNs on a specific mobile device, our compiler framework incorporates multiple optimizations such as operator fusion, operation replacement, and constant folding to effectively optimize the DNN computation graph. And we also adopt scheduling, nested parallelism, dense

Table 2. **Mobile acceleration comparison with MNN, TVM, and TensorFlow Lite for DNNs using mobile GPU on Samsung Galaxy S20.** The numbers represent the end-to-end inference latency in milliseconds (ms).

Model	MNN	TVM	TF-Lite	Ours (Dense)
VGG-16	139	221	307	**103**
ResNet-18	22.5	37.6	49.9	**19.8**
MobileNet-V2	13.3	20.5	24.3	**8.7**

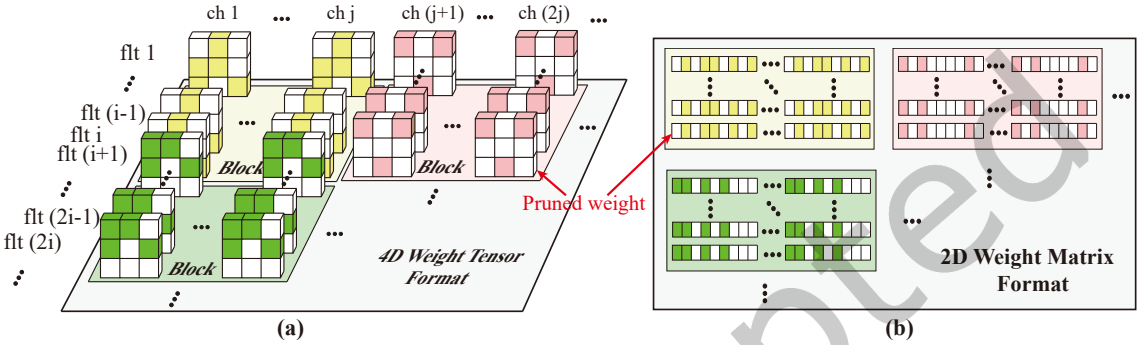

Fig. 6. Block-based pruning with 2D and 4D formats.

kernel reordering, and SIMD operation optimization techniques to enhance the execution efficiency. Auto-tuning is another key technique used in our framework, which can automatically obtain the best-suited runtime configuration for each unique case, e.g. tilling size, loop permutation, and data format.

To quantitatively demonstrate the effectiveness and the superiority of our proposed compiler optimization, we compare our framework with mobile-based solutions MNN, TVM, and TF-Lite using dense models without incorporating weight pruning. As shown in Table 2, the dense models of VGG-16, ResNet-18 and MobileNet-V2 on ImageNet dataset are evaluated. Even without incorporating weight pruning, mobile-based DNN executions can be improved compared to the state-of-the-art mobile-based solutions.

In our framework, besides the conventional unstructured pruning and coarse-grained filter/channel pruning, our optimized compiler also adds the flexibility in supporting a variety of pruning schemes including pattern-based pruning and our proposed block-based pruning. In fact, the unstructured and filter/channel pruning schemes are just extreme cases of our block-based pruning as shall be seen next. This is not available in the existing mobile-based solutions such as MNN, TVM, and TF-Lite.

4.2 Block-based Pruning

Inspired by the pattern-based pruning, our block-based pruning also adopts a fine-grained pruning strategy while maintaining a high regularity. Figure 6 (a) and (b) show our proposed block-based pruning in 4D weight tensor format and 2D weight matrix format, respectively. We divide entire weight matrix of a DNN layer into a number of equal-sized blocks, which contains the weights from j channels of i filters. Then we prune entire column(s) within each block to maintain the regular block shape. Moreover, our block-based pruning scheme requires pruning a group of weights at the same location of all filters and all channels within a block to leverage hardware parallelism from both memory and computation perspectives. Note that the conventional unstructured pruning and coarse-grained structured pruning can be considered the special cases of our proposed block-based pruning with a block size of 1×1 and the size of the whole weight matrix, respectively.

The key advantage of block-based pruning is that it can simultaneously achieve the advantages of unstructured pruning (high accuracy) and coarse-grained structured pruning (high hardware inference performance). The high accuracy is attributed to the structural flexibility. Since a finer pruning granularity is used in block-based pruning compared to coarse-grained structured pruning, higher structural flexibility is preserved when searching the desired pruning structures. On the other hand, the high hardware inference performance is attributed to the appropriate block size. Both FPGAs and mobile devices are limited by their computation resources. So even when the weight matrix is divided into multiple smaller blocks, each block can still be sufficient to exploit high hardware parallelism. Generally, a smaller block size provides higher structural flexibility, which achieves higher accuracy, but at the cost of reduced speed. On the contrary, a larger block size can better leverage the hardware parallelism to achieve higher acceleration, but it may cause severer accuracy loss. Thus, the block size is critical and should be specified depending on devices.

To determine an appropriate block size for mobile devices, we first determine the number of channels contained in each block by considering the computation resource of the device. To achieve high parallelism, we use the same number of channels for each block as the length of the vector registers in the mobile GPU. Then we determine the number of filters in each block by choosing the minimum number that can satisfy the inference speed requirement. A relatively larger block size is required for FPGAs than for mobile devices to fully leverage the hardware parallelism.

Note that the model with block-based sparsity can be obtained from offline training using different pruning algorithms such as group Lasso regularization [72] or Alternating Direction Methods of Multipliers (ADMM) [81]. There may be a difference in accuracy when adopting different pruning algorithms, but the energy efficiency will remain the same as long as the same pruning scheme and ratio are used. Since we are more focused on energy efficiency in this paper, we will not explore the performances of different pruning algorithms.

In this paper, we adopt the ADMM-based pruning method [81] to obtain the pruned models with our proposed block-based sparsity. In specific, we replace the unstructured sparsity constraints used in the ADMM-based method [81] with our proposed block-based sparsity constraints mentioned above. And an advantage of using the ADMM-based method is that it can automatically determine the desirable column and row pruning rates for each block given a predefined pruning rate for the entire layer.

4.3 Compiler Optimizations for Mobile Device

In this section, we introduce several critical compiler optimizations used in our framework, including general support for block-based pruning with different block sizes, a layer fusion mechanism to fuse different layers to shrink the intermediate result of each operator, and an auto-tuning strategy to further tune the performance for different models and different devices.

4.3.1 General Support for Block-based Pruning Scheme with Different Block Sizes. In general, a DNN model is constructed by stacking different layers (operators). In order to provide a layer-wise representation (LR) to describe each DNN layer, we design a domain-specific language (DSL) to represent the DNN model. During the DNN execution, the computational graph is built based on the DSL and further computational graph optimizations can also be applied to the DSL.

In our block-based pruning, we adopt the constraint that makes the same location of all filters and all channels within a block to be pruned. By pruning the same location in filters, these filters will skip reading the same input data, thus mitigating the memory pressure among the threads processing these filters. And pruning the same locations across channels within a block ensures that all of these channels share the same computation indices, thus eliminating the computation divergence among the threads processing the channels within each block. Such a design makes the block-based pruning more efficient for computation-intensive CONV layers.

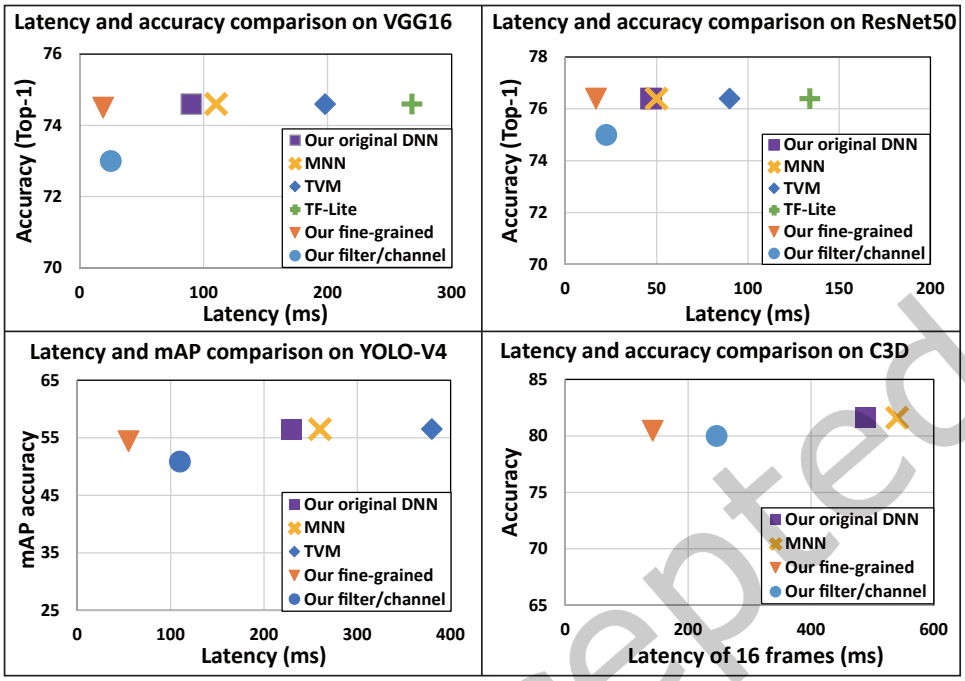

Fig. 7. Accuracy vs. inference latency on different DNN models and tasks. Results are measured on mobile GPU using 16-bit floating-point computations.

To compress the model storage, we store the sparse (non-zero) weights into a compact storage format. Inspired by the traditional CSR format, we use blocked compressed storage that can further compress indices array. The same optimization method can be used for the layers with different block sizes, thanks to the layer-wise representation.

4.3.2 Layer Fusion Mechanism. The layer fusion technique is used in our compiler optimization. It can fuse the computation operators in the computation graph to reduce the memory consumption of intermediate results and the number of operators, thereby effectively reducing the inference latency.

We identify all fusion candidates in a model based on two kinds of properties in the polynomial calculation: *computation laws* (i.e., associative property, commutative property, and distributive property) and *data access patterns*. Compared to the prior works using loop fusion [3, 5, 6], our fusion method is more aggressive without very expensive exploration. We only look for the fusion candidates that satisfy two constraints: (i) only explore the opportunities offered specifically because of the above properties, and (ii) only consider two cost metrics in the fusion, which are enlarging the overall computation to improve the CPU/GPU utilization and reducing the memory access to improve the memory performance.

4.3.3 Auto-tuning. In general, DNN executions involve many performance-critical tuning parameters, such as the data placement on GPU memory, matrix tiling sizes, loop unrolling factors, etc. It is hard to identify the best-suited configuration of these parameters manually. In our work, our compiler incorporates auto-tuning approaches as other DNN inference frameworks like TVM. To facilitate an efficient auto-tuning process, we utilize the Genetic Algorithm to explore the best configuration automatically, which allows starting parameter search

Table 3. **Energy efficiency comparison with MNN, TVM, and TF-Lite for DNN executions using the mobile GPU on Samsung Galaxy S10.**

Model	MNN	TVM	TF-Lite	Ours
VGG-16	4.3	2.7	1.8	**33**
ResNet-18	27.2	15.8	11.8	**65.5**
MobileNet-V2	44.8	29.6	26.2	**146**

with initializing an arbitrary number of chromosomes, better exploring the parallelism. These optimizations, together with the layer-fusion optimization, render our framework to outperform other acceleration frameworks.

4.4 Optimization Results for Mobile Device

By incorporating weight pruning with our compiler optimizations, our framework can further boost the mobile-based DNN executions. As shown in Figure 7, we evaluate our framework on VGG16 and ResNet50 for image classification task, YOLO-V4 for object detection task, and C3D for 3D activity detection task, on ImageNet [15], MS-COCO [43], and UCF-101 [68] dataset, respectively. We compare the results of our framework with representative mobile-based solution MNN, TVM, and TF-Lite. To evaluate the effectiveness of our proposed block-based pruning scheme separately, we also compare our block-based pruning results to the dense models and the models pruned by coarse-grained filter/channel pruning, while all the results are optimized by our compiler optimization.

From Figure 7, we can observe that by adopting our compiler optimization and block-based pruning, a much lower inference latency (faster inference speed) can be consistently achieved than MNN, TVM, and TF-Lite. Our framework has no accuracy loss on VGG16 and ResNet50, and a slight accuracy degradation on YOLO-V4 and C3D.

We also show a clear advantage of our proposed block-based pruning schemes in comparison with coarse-grained structured (filter/channel) pruning in terms of both accuracy and latency. Benefiting from the finer granularity, higher structural flexibility is preserved in our block-based pruning scheme for searching a more desired pruning structure, eventually leading to a higher accuracy. With higher accuracy maintained, a much higher compression rate can be achieved by using block-based pruning than coarse-grained structured pruning, resulting in a lower inference latency.

Besides, we compare the energy efficiency of our proposed framework to FPGA-based solutions and mobile-based solutions. We use frames per second per watt (FPS/W), which accounts for the end-to-end DNN inference speed, to demonstrate the improvement of energy efficiency achieved by our compiler optimizations and block-based pruning. This serves as a better metric for evaluating the result of weight pruning as the total computation is reduced. As shown in Figure 8, with similar accuracy, our proposed framework outperforms the best mobile-based solution MNN by 2.36×, 3.26×, and 7.71× in terms of FPS/W on ResNet18, MobileNetV2, and VGG16, respectively.

Other than using Samsung Galaxy S20 smartphone, we also evaluate the energy efficiency performance of Samsung Galaxy S10 smartphone. The Samsung Galaxy S10 is less powerful than Samsung Galaxy S20, which has Qualcomm Snapdragon 855 chipset and Adreno 640 mobile GPU. In this way, we can evaluate the transferability of our framework and the energy efficiency comparison to the FPGA-based designs. The energy efficiency results using Samsung Galaxy S10 are shown in Table 3. It can be observed that our framework still consistently outperforms MNN, TVM, TF-Lite, and the FPGA-based designs.

4.5 Optimization Results for FPGA

A key challenge in the FPGA acceleration under our block-based sparsity is the workload imbalance issue. Because block-based pruning may lead to different degrees of sparsity between different blocks and may not provide enough inner-block parallelism to fully exploit the large volume of computing resources on an FPGA.

The workload sharing technique [67] can be used to address the workload imbalance issue. With the help of compiler-level code generation, the workload can be shared in an online and adaptive manner.

Figure 9 shows the speedup and accuracy results of our block-based pruning adopted on FPGA under different block sizes. The speedup is normalized to the dense (unpruned) model, with weight/activation in 8 bits. We can observe that, when using the block size of 16×16 and 32×32, our block-based pruning achieves much higher speedup compared to unstructured pruning while having a minor accuracy drop. This indicates our proposed block-based pruning can effectively accelerate DNN execution on FPGAs, while maintaining accuracy. Larger block size is required to fully exploit hardware parallelism on FPGAs to achieve a higher speedup, but it will also lead to a larger accuracy drop.

In a nutshell, our proposed block-based pruning can benefit both mobile-based and FPGA-based accelerations. It is important to be noted that, when both mobile devices and FPGAs have been applied with weight pruning, the hardware performance of both can be enhanced while maintaining high accuracy. It is still difficult for FPGAs to mitigate the gap and achieve comparable energy efficiency compared to mobile devices under the same accuracy.

5 CONCLUSION

In this paper, we conduct a thorough analysis of FPGAs and mobile devices under both baseline executions as well as optimized executions where state-of-the-art optimizations are applied. We observe that, for DNN executions, it is very difficult for FPGAs to beat mobile devices when considering both energy-efficiency and model accuracy. To further explore the benefit of weight pruning, we propose a novel hardware friendly pruning strategy that can be effectively applied to both mobile devices and FPGAs, along with compiler-assisted acceleration framework.

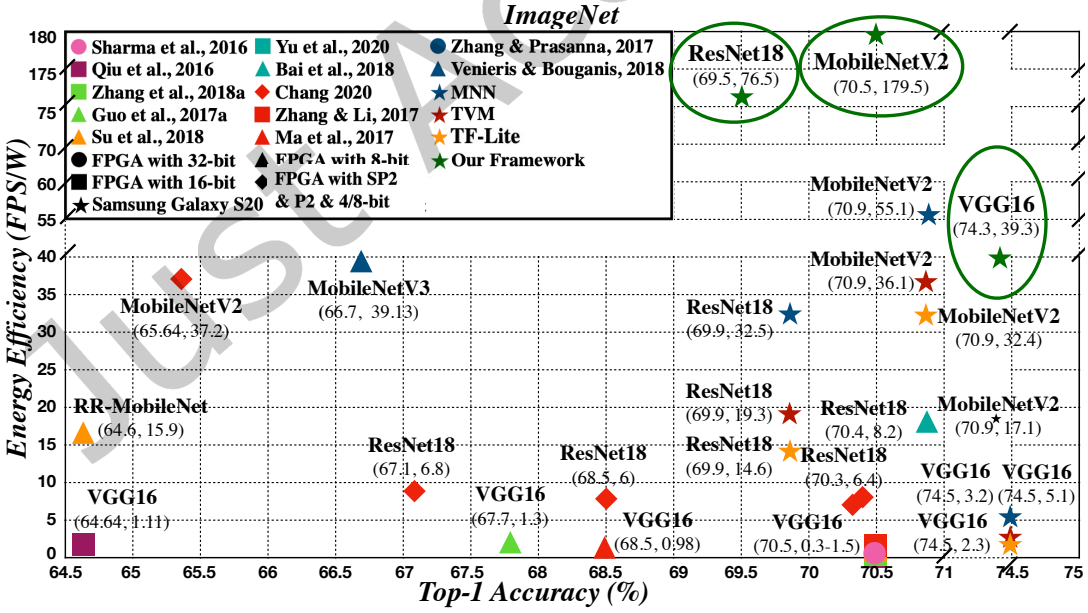

Fig. 8. Energy efficiency (FPS/W) vs. accuracy on Mobile Phone & FPGA on DNNs with ImageNet dataset.

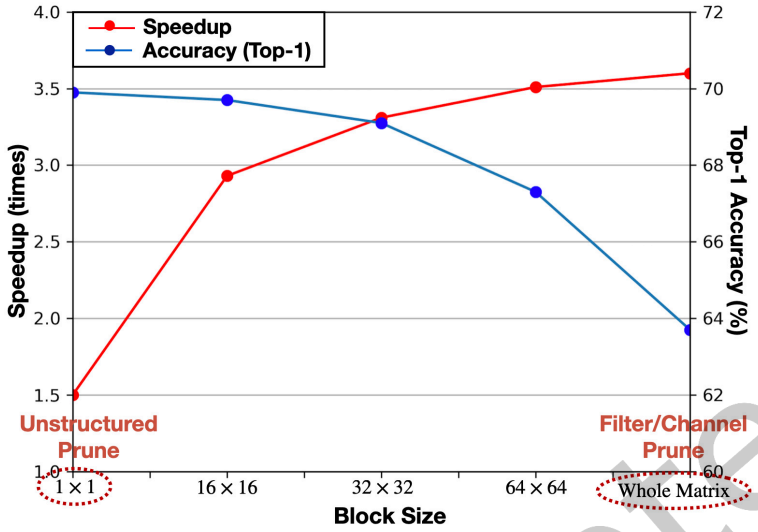

Fig. 9. Top-1 accuracy and FPGA inference latency of ResNet-18 on ImageNet dataset with different block sizes, under a uniform pruning rate of 6× for all layers. Results are measured on the Xilinx ZCU102 FPGA board.

ACKNOWLEDGMENTS

This research is partially funded by National Science Foundation CCF-1919117, CNS-1909172, CCF-2047516 (CAREER), and CCF-1901378, and Jeffress Trust Awards in Interdisciplinary Research to William & Mary. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or Thomas F. and Kate Miller Jeffress Memorial Trust.

REFERENCES

[1] [n.d.]. https://www.tensorflow.org/mobile/tflite/.
[2] [n.d.]. https://www.qualcomm.com/products/snapdragon-865-plus-5g-mobile-platform.
[3] Arash Ashari, Shirish Tatikonda, Matthias Boehm, Berthold Reinwald, Keith Campbell, John Keenleyside, and P Sadayappan. 2015. On optimizing machine learning workloads via kernel fusion. *ACM SIGPLAN Notices* 50, 8 (2015), 173–182.
[4] Lin Bai, Yiming Zhao, and Xinming Huang. 2018. A CNN accelerator on FPGA using depthwise separable convolution. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 10 (2018), 1415–1419.
[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM review* 59, 1 (2017), 65–98.
[6] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Alexandre V Evfimievski, and Prithviraj Sen. 2018. On optimizing operator fusion plans for large-scale machine learning in systemml. *arXiv preprint arXiv:1801.00829* (2018).
[7] Sung-En Chang, Yanyu Li, Mengshu Sun, Weiwen Jiang, Runbin Shi, Xue Lin, and Yanzhi Wang. 2020. MSP: An FPGA-Specific Mixed-Scheme, Multi-Precision Deep Neural Network Quantization Framework. *arXiv preprint arXiv:2009.07460* (2020).
[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.
[9] Gong Cheng, Lu Ye, Li Tao, Zhang Xiaofan, Hao Cong, Chen Deming, and Chen Yao. 2019. μL2Q: An Ultra-Low Loss Quantization Method for DNN. *The 2019 International Joint Conference on Neural Networks (IJCNN)* (2019).
[10] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).
[11] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. 2018. Understanding Performance Differences of FPGAs and GPUs: (Abtract Only). In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*

(Monterey, CALIFORNIA, USA) *(FPGA '18)*. Association for Computing Machinery, New York, NY, USA, 288.
[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems (NeurIPS)*. 3123–3131.
[13] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).
[14] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. 2017. NeST: a neural network synthesis tool based on a grow-and-prune paradigm. *arXiv preprint arXiv:1711.02017* (2017).
[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.
[16] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. 2017. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 395–408.
[17] Caiwen Ding, Shuo Wang, Ning Liu, Kaidi Xu, Yanzhi Wang, and Yun Liang. 2019. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 33–42.
[18] Xuanyi Dong and Yi Yang. 2019. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*. 759–770.
[19] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *International Conference on Learning Representations (ICLR)* (2019).
[20] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n.d.]. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
[21] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and TN Vijaykumar. 2019. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 151–165.
[22] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 4852–4861.
[23] Kaiyuan Guo, Song Han, Song Yao, Yu Wang, Yuan Xie, and Huazhong Yang. 2017. Software-Hardware Codesign for Efficient Neural Network Acceleration. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 18–25.
[24] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2017. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 37, 1 (2017), 35–47.
[25] Peng Guo, Hong Ma, Ruizhi Chen, Pin Li, Shaolin Xie, and Donglin Wang. 2018. Fbna: A fully binarized neural network accelerator. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 51–513.
[26] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*. 1379–1387.
[27] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.
[28] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
[29] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4340–4349.
[30] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 1398–1406.
[31] Zhezhi He and Deliang Fan. 2019. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11438–11446.
[32] Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. 2019. MnnFast: a fast and scalable system architecture for memory-augmented neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*. 250–263.
[33] Weiwen Jiang, Edwin H-M Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. 2019. Achieving super-linear speedup across multi-fpga for real-time dnn inference. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–23.
[34] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *arXiv preprint arXiv:2007.09087* (2020).
[35] Weiwen Jiang, Lei Yang, Edwin H-M Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/Software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and*

Systems (2020).
[36] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. 2019. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
[37] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lyu, and Zhihua Wu. 2020. MNN: A Universal and Efficient Inference Engine. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.). Vol. 2. 1–13.
[38] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4350–4359.
[39] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. 2018. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*.
[40] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).
[41] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. 2019. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3977–3986.
[42] Yuhang Li, Xin Dong, and Wei Wang. 2020. Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks. In *International Conference on Learning Representations (ICLR)*.
[43] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.
[44] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*. 345–353.
[45] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2019. AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. *arXiv preprint arXiv:1907.03141* (2019).
[46] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).
[47] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. 2019. On neural architecture search for resource-constrained hardware platforms. *arXiv preprint arXiv:1911.00105* (2019).
[48] Cheng Luo, Wei Cao, Lingli Wang, and Philip HW Leong. 2019. Rna: An accurate residual network accelerator for quantized and reconstructed deep neural networks. *IEICE Transactions on Information and Systems* 102, 5 (2019), 1037–1045.
[49] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*. 5058–5066.
[50] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. 2020. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *Thirty-Fourth AAAI conference on artificial intelligence (AAAI)*.
[51] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. 2020. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. *AAAI* (2020).
[52] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 45–54.
[53] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922* (2017).
[54] Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).
[55] Hiroki Nakahara, Tomoya Fujii, and Shimpei Sato. 2017. A fully connected layer elimination for a binarized convolutional neural network on an FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.
[56] Hiroki Nakahara, Masayuki Shimoda, and Shimpei Sato. 2018. A Tri-State Weight Convolutional Neural Network for an FPGA: Applied to YOLOv2 Object Detector. In *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 298–301.
[57] Hiroki Nakahara, Haruyoshi Yonekawa, Tomoya Fujii, and Shimpei Sato. 2018. A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga. In *Proceedings of the 2018 ACM/SIGDA International Symposium on field-programmable gate arrays*. 31–40.
[58] Hiroki Nakahara, Haruyoshi Yonekawa, Tsutomu Sasao, Hisashi Iwamoto, and Masato Motomura. 2016. A memory-based realization of a binarized deep convolutional neural network. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 277–280.
[59] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim, and Hyuk-Jae Lee. 2019. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 8 (2019), 1861–1873.

[60] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[61] Yue Niu, Rajgopal Kannan, Ajitesh Srivastava, and Viktor Prasanna. 2020. Reuse Kernels or Activations? A Flexible Dataflow for Low-latency Spectral CNN Acceleration. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 266–276.

[62] Thomas B Preußer, Giulio Gambardella, Nicholas Fraser, and Michaela Blott. 2018. Inference of quantized neural networks on heterogeneous all-programmable devices. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 833–838.

[63] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees A. Vissers, Joseph Zambreno, and Phillip H. Jones. 2019. Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels. In *15th IEEE International Conference on Embedded Software and Systems, ICESS 2019, Las Vegas, NV, USA, June 2-3, 2019*. IEEE, 1–8. https://doi.org/10.1109/ICESS.2019.8782524

[64] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 26–35.

[65] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision (ECCV)*. Springer, 525–542.

[66] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 1–13.

[67] Runbin Shi, Peiyan Dong, Tong Geng, Yuhao Ding, Xiaolong Ma, Hayden K-H So, Martin Herbordt, Ang Li, and Yanzhi Wang. 2020. CSB-RNN: A Faster-than-Realtime RNN Acceleration Framework with Compressed Structured Blocks. *arXiv preprint arXiv:2005.05758* (2020).

[68] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. 2012. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* (2012).

[69] Jiang Su, Julian Faraone, Junyi Liu, Yiren Zhao, David B Thomas, Philip HW Leong, and Peter YK Cheung. 2018. Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet classification. In *International Symposium on Applied Reconfigurable Computing*. Springer, 16–28.

[70] Stylianos I Venieris and Christos-Savvas Bouganis. 2018. fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. *IEEE transactions on neural networks and learning systems* 30, 2 (2018), 326–342.

[71] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. 2018. Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 163–1636.

[72] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. 2074–2082.

[73] Jincheng Yu, Kaiyuan Guo, Yiming Hu, Xuefei Ning, Jiantao Qiu, Huizi Mao, Song Yao, Tianqi Tang, Boxun Li, Yu Wang, and Huazhong Yang. 2018. Real-time object detection towards high power efficiency. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 704–708.

[74] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9194–9203.

[75] Yunxuan Yu, Tiandong Zhao, Kun Wang, and Lei He. 2020. Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 122–132.

[76] Chi Zhang and Viktor Prasanna. 2017. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 35–44.

[77] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2018. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 11 (2018), 2072–2085.

[78] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 365–382.

[79] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. 2019. Eager pruning: algorithm and architecture support for fast training of deep neural networks. In *Proceedins of the 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 292–303.

[80] Jialiang Zhang and Jing Li. 2017. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 25–34.

[81] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. *arXiv preprint arXiv:1804.03294* (2018).

[82] Ruizhe Zhao, Xinyu Niu, Yajie Wu, Wayne Luk, and Qiang Liu. 2017. Optimizing CNN-based object detection algorithms on embedded FPGA platforms. In *International Symposium on Applied Reconfigurable Computing*. Springer, 255–267.
[83] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights. In *International Conference on Learning Representations (ICLR)*.
[84] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).
[85] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2017. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*.