CoDeNet: Algorithm-hardware Co-design for Deformable Convolution

Zhen Dong*, Dequan Wang*, Qijing Huang*, Yizhao Gao[1], Yaohui Cai[2]
Bichen Wu, Kurt Keutzer, John Wawrzynek
University of California, Berkeley, [1]University of Chinese Academy of Science, [2]Peking University
{zhendong,dqwang,qijing.huang,bichen,keutzer,johnw}@eecs.berkeley.edu
gaoyizhao16@mails.ucas.ac.cn,caiyaohui@pku.edu.cn

ABSTRACT

Deploying deep learning models on embedded systems for computer vision tasks has been challenging due to limited compute resources and strict energy budgets. The majority of existing work focuses on accelerating image classification, while other fundamental vision problems, such as object detection, have not been adequately addressed. Compared with image classification, detection problems are more sensitive to the spatial variance of objects, and therefore, require specialized convolutions to aggregate spatial information. To address this, recent work proposes dynamic deformable convolution to augment regular convolutions. Regular convolutions process a fixed grid of pixels across all the spatial locations in an image, while dynamic deformable convolution may access arbitrary pixels in the image and the access pattern is input-dependent and varies per spatial location. These properties lead to inefficient memory accesses of inputs with existing hardware. In this work, we first investigate the overhead of the deformable convolution on embedded FPGA SoCs, and introduce a depthwise deformable convolution to reduce the total number of operations required. We then show the speed-accuracy tradeoffs for a set of algorithm modifications including irregular-access versus limited-range and fixed-shape. We evaluate these algorithmic changes with corresponding hardware optimizations. Results show a 1.36× and 9.76× speedup respectively for the full and depthwise deformable convolution on the embedded FPGA accelerator with minor accuracy loss on the object detection task. We then co-design an efficient network CoDeNet with the modified deformable convolution for object detection and quantize the network to 4-bit weights and 8-bit activations. Results show that our designs lie on the pareto-optimal front of the latency-accuracy tradeoff for the object detection task on embedded FPGAs.

1 INTRODUCTION

Convolution is widely adopted in different neural network architecture designs for various object recognition tasks. Many hardware accelerators have been developed to improve the speed and power performance of the compute-intensive convolutional kernels. While the use of convolution kernels for computer vision is well-established, researchers have been constantly proposing new operations and new network designs, to increase the model capability and achieve better speed-accuracy trade-off for various tasks. Deformable convolution [5][43] is one of the novel operations that leads to the state-of-the-art accuracy for object recognition with more effective usage of parameters. Many neural network designs with top accuracy for object detection on the COCO dataset [20] use deformable convolution in their design, including the 1st-ranked model to date [37]. Differing from the conventional convolutions with fixed geometric structure, deformable convolution samples inputs from variable offsets generated based on the input features during inference. There are two advantages it provides compared to conventional convolutions: *variable sampling scales* and *variable sampling geometry*. The range for sampling at each different point varies, allowing the network to capture objects of different scales. The geometry of the sample points is not fixed, allowing the network to capture objects of different shapes. Several previous studies [21][3][18][41] have also shown that deformable convolution design lies on the pareto front of the speed-accuracy tradeoff for object detection on GPUs.

There are several challenges in supporting deformable convolution on embedded hardware accelerators. (i) The memory accesses for the input feature maps are irregular as they depend on the dynamically-generated offsets. Many existing accelerators' instruction set architecture and the control logic are insufficient in supporting the random memory access patterns. In addition, the less contiguous memory access patterns limit the length of bursting memory accesses and incur more memory requests. (ii) There is less spatial reuse for the input features. Due to the variable filter offsets, the loaded input pixel for the current output pixel can no longer be reused by its neighboring output pixels. This can significantly affect the performance of the accelerators designed for output-stationary or row-stationary dataflow which leverages input reuse. (iii) There is an increased memory bandwidth requirement for loading the variable offsets.

To address these challenges, we adopt an algorithm-hardware co-design approach and study the accuracy-efficiency tradeoffs for each algorithmic modification on an embedded field-programmable gate array (FPGA) with limited hardware resources. As a programmable platform, FPGA lends itself to accelerating fast-evolving deep learning algorithms. Compared to other general-purpose platforms at the edge, it has higher power efficiency and better low-batch inference performance. In addition, timely and efficient hardware support for novel operations can be developed on FPGAs in weeks with high-level design tools.

In this work, we propose the following modifications to the deformable convolution operation to make it more hardware friendly:

(1) Limit the adaptive offsets to a fixed range to allow buffering of inputs and exploit full input reuse
(2) Constrain the arbitrary offset displacements into a square shape to reduce the overhead from loading the offsets and to enable parallel accesses to on-chip memory

*Equal Contribution.

(3) Round the offset displacements into integers and remove the fractional, bilinear interpolation operation for calculating the final sampling value
(4) Use depth-wise convolution to reduce the total number of Multiply-Accumulate operations (MACs).

We evaluate each modification on an FPGA System-on-Chip (SoC) that includes both an FPGA fabric and a hardened CPU core. We leverage the shared last-level cache (LLC) included in its full hardened processor system to efficiently exploit the locality of deformable convolution with data-dependent memory access patterns. We then optimize the hardware based on each algorithm modification to demonstrate its advantage in efficiency over the original operation. With these proposed algorithm modifications, we devise a line-buffer design to efficiently support our optimized depthwise deformable convolutional operation. To demonstrate the full capability of the co-designed operation, we also design an efficient deep neural network (DNN) model CoDeNet for object detection using ShuffleNetV2 [23] as the feature extractor. We quantize the network to 4-bit weights and 8-bit activations with a symmetric uniform quantizer using the block-wise quantization-aware fine-tuning process [7].

Our contributions include:

(1) Co-design a novel depthwise deformable convolution with hardware-friendly modifications
(2) Optimize the hardware design for each algorithm modification and demonstrate the accuracy and hardware efficiency trade-off for each algorithmic modification we propose
(3) Integrate the proposed depthwise deformable convolution in an efficient deep neural network for object detection and quantize the model to low-precision.
(4) Implement a hardware accelerator targeting the new network design on an FPGA SoC

The rest of the paper is organized as follows: Section 2 gives an introduction to the deformable convolution; Section 3 provides an ablation study for the operation co-design; Section 4 describes the end-to-end object detection system we design with the modified operation; Section 5 shows our final performance results and we conclude the paper in Section 6.

2 BACKGROUND

2.1 Object Detection

Object detection is a more challenging task than image classification as it performs object localization in addition to object classification and requires spatial-variant dense prediction. The existing solutions can be categorized into two-stage and one-stage approaches. Two-stage algorithms need to first propose a set of regions of interest and then perform object classification on the selected region candidates. Faster R-CNN [29] introduces a Region Proposal Network (RPN) used for hypothesizing object locations in two-stage algorithms. RPN is a fully convolutional sub-network that shares features with the detection network to reduce the cost of region proposal generation. One-stage algorithms skip the region proposal stage and directly run detection over a dense sampling of possible regions. Single Shot MultiBox Detector (SSD) [22] leverages pyramidal feature hierarchy in the feature extraction network to efficiently encode objects within various sizes. You Only Look Once (YOLO) [27][28] is a one-stage algorithm using fully convolutional network(FCN). The algorithm divides the input image into a feature grid. Each cell in the grid predicts bounding boxes with location information, confidence scores indicating the probability of an object in these boxes, and the conditional probability of the object class.

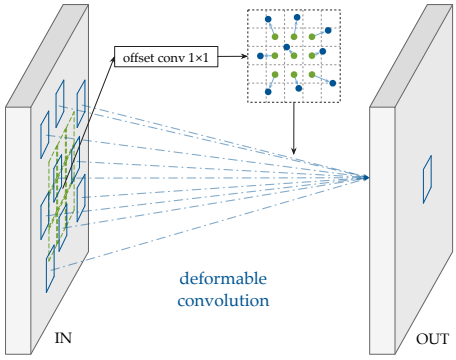


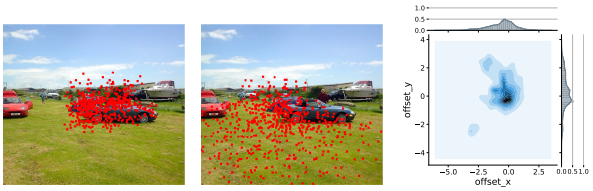
Figure 1: Deformable convolution with variable displacement offset generation. We first utilize a 1×1 convolutional layer to generate the sampling displacement. Then the filter would aggregate the corresponding features in a convolutional way, weighted by the kernel weight.

We use the recent CenterNet [41] design for the detector in our work. It is a simple anchor-free design with better pareto-optimal front for the speed-accuracy tradeoff compared to the other concurrent works [8][15][16][42]. Most of the anchor-free detectors still need to use the Non Maximum Suppression (NMS) mechanism to remove the duplicated predictions as their training procedure assigns multiple positive samples to the foreground objects. On the contrary, CenterNet directly generates the center point for each object without requiring any post-processing, which could be seen as the simplest anchor-free design.

As for the evaluation metrics for object detection, a common practice is to use the average precision (AP) and intersection over union (IoU). AP computes the average precision value achieved with different recall values. A precision value is defined as $\frac{\text{true positive}}{\text{true positive+false positive}}$, and a recall value is defined as $\frac{\text{true positive}}{\text{true positive+false negative}}$. IoU is defined as the overlap between the area of the boxes intersection over the area of the boxes union. The default evaluation metric for VOC dataset [9] is AP50, which indicates that the prediction would be seen as correct if the corresponding IoU $\geq$ 0.5. The main metric for COCO is the mean of the average precisions at IoU from 0.5 to 0.95 with a step size 0.05.

2.2 Deformable Convolution

Compared to image classification, the major challenge of object detection is to capture geometric variations of each object, such as scale, pose, viewpoint, and part deformation, in a spatial variant way within the image. State-of-the-art approaches [3][18][21][31][41] to address the challenge is to utilize deformable convolution [5][43]. As demonstrated in Figure 1, deformable convolution samples the input feature map using the offsets dynamically predicted from the same input feature map. The convolution layer for generating the



(a) car (b) lawn (c) distribution

Figure 2: Example for deformable convolution sampling locations and offset range distribution. (a) the sampling locations for the car as an active unit. (b) the sampling locations for lawn in the the background. (c) distribution of the deformable offsets for the example image.

offsets is jointly trained with the rest of the network via standard backpropagation in an end-to-end manner. Thus, the gradient updates not only the weights but also the sampling locations for the convolution, allowing more flexible and adaptive sampling. Unlike the conventional convolution with fixed geometry, its receptive fields can be of various shapes to capture objects with different scales, aspect ratios, and rotation angles. Besides, deformable convolution is both spatial-variant and input-adaptive. In other words, its sampling patterns and offsets vary for different output pixels in the same input feature map and also vary across different input feature maps. In Figure 2(a)(b), we show how the sampling locations (red dots) change with the different active units (the object with a green dot on it). Albeit the operation augments and enhances the capability of the existing convolution for object detection, its dynamic nature poses extra challenges to the existing hardware.

2.3 Algorithm-hardware Co-design

Many prior acceleration works [24][25][10][38][33] have demonstrated the effectiveness of the co-design methodology for the deployment of real-time object detection on FPGAs. [24] customizes SSD300 [22] by replacing operations, such as dilated convolutions, normalization, and convolutions with larger stride, with more efficiently supported ones on FPGAs. [25] adapts YOLOv2 [28] by introducing a binarized network as the backbone for feature extraction to leverage the low-precision support of FPGA. Meanwhile, the FINN-R framework [2] further explores the benefits of integrating quantized neural networks (QNN) into Yolo-based object detection systems. A real-time object detection for live video streaming system [26] enables is then developed with the FINN-based QNNs. [10] devised an automatic co-design flow on embedded FPGAs for the DJI-UAV [34] dataset with 95 categories targeting unmanned aerial vehicles. The flow first constructs DNN basic building blocks called bundles, estimates their corresponding latency and cost on hardware, and selects the ones on the pareto front for latency and resources trade-off. Then it starts a two-phase DNN evaluation to search for the bundles on the pareto front of the accuracy-latency trade-off and then fine-tune the design of the selected bundles. SkyNet [38] searched by this co-design flow achieves the best performance (based on a combination of throughput, power, and detection accuracy) on embedded GPUs and FPGAs.

2.4 Quantization

Quantization [40][13][36][7] is a promising approach for efficient deployment of neural network models on the embedded devices. It alleviates the memory bottleneck by representing weights in neural network models with ultra-low precision such as 4-bits. Moreover, instead of floating-point matrix multiplication, quantizing both weights and activations enables the use of low-precision integer arithmetics, which enables significant acceleration for the inference. However, directly performing aggressive layer-wise quantization can lead to significant accuracy degradation [14]. Many prior works have attempted to address this accuracy gap with various techniques, such as non-uniform learnable quantizer [36], mixed-precision quantization [6], progressive fine-tuning [39] as well as group-wise [32] and channel-wise quantization [14]. Although these methods can better preserve the accuracy of the pre-trained model, they also increase the complexity of hardware implementation and may cause non-negligible overhead on both latency and memory usage. Consequently, it is crucial to carefully consider the trade-off between accuracy and hardware efficiency when deploying a quantized model on the edge devices. Quantization performance also has a strong relation to the network architecture and the target task. [14] shows that compact models are more difficult to quantize. And in contrast to image classification, object detection is a more challenging task for ultra-low precision quantization since it requires accurate localization of specific objects in an image. Even with quantization-aware fine-tuning, quantizing the detection models with naive quantization schemes can cause around 10% AP degradation on the COCO dataset [17]. In [17], a quantization scheme specifically designed for object detection is presented, leading to 3.1 AP degradation on their 4-bit RetinaNet [19].

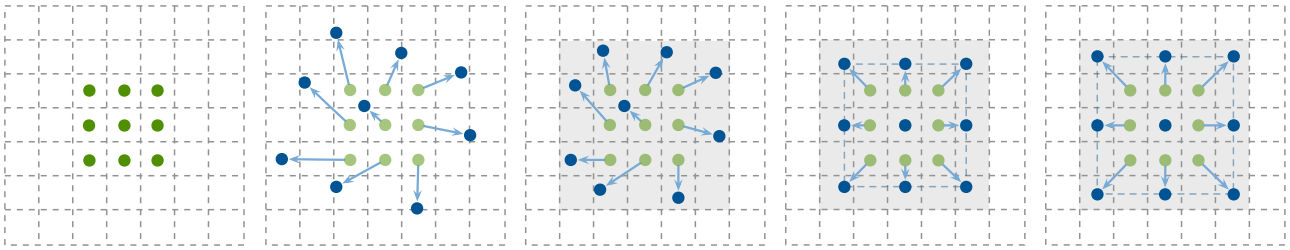
3 DEFORMABLE OPERATION CO-DESIGN

It is challenging to provide efficient support for the original deformable convolution on off-the-shelf hardware accelerators due to: (i) the limited reuse of input features, (ii) the dynamic and irregular input-dependent memory access patterns, (iii) the computation overhead from the fractional bilinear interpolation, (iv) the memory overhead of the deformable offsets. In this work, we perform a series of modifications to the algorithm to make the operation more hardware-friendly. A comprehensive ablation study is done to demonstrate the impact of each algorithmic modification on accuracy. We perform our study with standard object detection benchmarks, VOC and COCO. We then design a specialized hardware engine optimized for each algorithmic modification on FPGA and show the performance improvement on FPGA from each modification. We demonstrate the accuracy and hardware efficiency trade-off for each modification we propose.

We will be using the following notations in the paper: n - batch size, h - height, w - width, ic - input channel size, oc - output channel size, k - kernel size, Δp - offsets.

3.1 Algorithm Modifications

We choose average precision (AP) as the main metric for benchmarking object detection performance on VOC and COCO datasets. ShuffleNet V2 [23] is used as the feature extractor in all experiments. As for decoder, we follow the practice of CenterNet [41]



(a) normal (b) deform (c) bound (d) square (e) round

Figure 3: Major algorithm modifications for deformable convolution operational co-design. (a) is the default 3×3 convolutional filter. (b) is the original deformable convolution with unconstrained non-integer offsets. (c) sets an upper bound to the offsets. (d) limits the geometry to a square shape. (e) shows that the predicted offsets are rounded to integers.

Operation	Depthwise	Bound	Square	VOC			COCO					
				AP	AP50	AP75	AP	AP50	AP75	APs	APm	APl
3×3				39.2	60.8	41.2	21.4	36.5	21.5	7.3	24.1	33.0
3×3	✓			39.1	60.9	40.9	19.8	34.3	19.7	6.3	22.6	31.5
5×5	✓			40.6	62.4	42.6	21.3	36.4	21.3	6.7	23.7	34.2
7×7	✓			41.9	63.8	43.8	21.7	37.2	21.5	6.9	24.0	35.2
9×9	✓			42.3	64.8	44.3	22.2	37.8	22.1	7.0	24.3	35.4
deform	✓			42.9	64.4	45.7	23.0	38.4	23.3	6.9	24.4	37.8
deform	✓	✓		41.0	63.0	42.9	21.3	36.4	21.1	7.2	23.6	34.4
deform	✓	✓	✓	41.1	63.1	43.7	21.5	36.8	21.5	6.5	23.7	34.8
deform*	✓	✓	✓	43.4	65.7	45.7	24.2	39.8	24.7	8.9	25.8	37.5

Table 1: Ablation study of operation choices for object detection on VOC and COCO. The upper part shows the baselines with various kernel sizes, from 3×3 to 9×9. The lower part shows the comparison of different design choices on deformable convolution.

and use the stack of deformable convolution, nearest 2× upsample, and ReLU activation layers. Table 1 lists the modifications we make to the original deformable convolution as well as a comparison between deformable convolution and convolution with large kernels. From the comparison, we see that deformable convolution achieves higher accuracy on Pascal VOC compared to convolution with 9×9 kernel (42.9 vs 42.3) whiling requiring $\frac{9\times9}{3\times3} = 9\times$ fewer MACs and weight parameters. We perform several modifications to further improve its efficiency and discuss them in this section.

Depthwise Convolution We first replace the full 3×3 deformable convolutions with 3×3 depthwise deformable convolutions and 1×1 convolutions, similar to the depthwise separable convolution practice in Xception [4]. Such modification makes the whole network more uniform and smaller, so the weights of the deformable convolution can be all buffered on-chip for maximal reuse.

Bounded Range Our next algorithmic modification to facilitate efficient hardware acceleration is to restrict the offsets to a positive range. Such constraint limits the size of the working set of feature maps, so that a pre-defined fixed-size buffer can be added to the hardware, in order to further exploit the temporal and spatial locality of the inputs. Assume a uniform distribution for the generated offsets in a 3×3 convolution kernel with stride 1, each pixel is expected to be used nine times. If all inputs within the range can be stored in the buffer, all except the first access to the same address will be from on-chip memory with $1 \sim 3$ cycle latency. We impose this constraint during training by adding a *clipping* operation after the offset generation layer to truncate offsets that are smaller than 0 or larger than N, so all offsets $\Delta p_x, \Delta p_y \in [0, N]$. Table 1 shows that setting the bound N to 7 results in 1.9 and 1.7 AP degradation on VOC and COCO respectively.

Square Shape Another obstacle to efficiently supporting the deformable convolution is its irregular data access patterns, which leads to serialized memory accesses to multi-banked on-chip memory. To address this issue, we further constrain the offsets to be on the edges of a square. Instead of using $3 \times 3 \times 2 = 18$ numbers to represent the Δp_x and Δp_y offsets for all nine samples, only one number Δp_d, representing the distance from the center to the sides of the square, needs to be learned. This is similar to a dilated convolution with spatial-variant adaptive dilation factors. Adding this modification leads to 0.1 and 0.2 AP decrease on VOC and COCO.

Rounded Offsets In the original deformable design, the generated offsets are typically fractional and a bilinear interpolation needs to be performed to produce the target sampling value. Bilinear interpolation calculates a weighted average of the neighboring pixels for a fractional offset based on its distance to the neighboring pixels. It introduces at least six multiplications to the sampling process of each input, which is a significant increase ($6 \times h \times w \times ic$) to the total FLOPs. We thus round the offsets to be integers during inference to reduce the total computation. The dynamically-generated offsets are thus rounded to integers. In practice, we round the generated offset during the quantization step.

Operation	Deform	Bound	Square	Without LLC		With LLC	
				Latency	GOPs	Latency	GOPs
				43.1	112.0	41.6	116.2
default	✓			59.0	81.8	42.7	113.1
3×3 conv	✓	✓		43.4	111.5	41.8	115.5
	✓	✓	✓	43.4	111.5	41.8	115.6
				1.9	9.7	2.0	9.6
depthwise	✓			20.5	0.9	17.8	1.1
3×3 conv	✓	✓		3.0	6.2	3.4	5.5
	✓	✓	✓	2.1	9.2	2.3	8.2

Table 2: Co-designed hardware performance comparison.

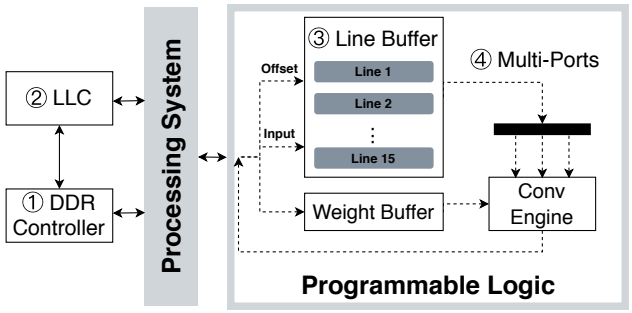


Figure 4: Hardare engine for deformable convolution

As shown in Table 1, together with the modifications above, our co-designed deformable convolution achieves 41.1 and 21.5 AP on VOC and COCO respectively, which is 1.8 and 1.5 lower than the original depthwise deformable convolution. Note that the accuracy of the modified deformable convolution still achieves higher accuracy compared to the large 5×5 kernel, while requiring $\frac{3\times3}{5\times5} = 36\%$ less MACs and parameters. The *deform** entry shows an improved design of the network with the deformable convolution used in the feature exactor and the addition of FPN. Its accuracy is higher than MobileNetV2+SSD [30] but with more compact model design.

3.2 Hardware Optimizations

Many hardware optimization opportunities are exposed after we perform the aforementioned modifications to deformable convolution. We implement a hardware deformable convolution engine on FPGA SoC as shown in Figure 4 and tailor the hardware engine to each algorithm modification. The experiments are run on the Ultra96 board featuring a Xilinx Zynq XCZU3EG UltraScale+ MPSoC platform. The accelerator logic accesses the 1MB 16-way set-associative LLC through the Accelerator Coherency Port (ACP). The data cache uses a pseudo-random replacement policy. Table 2 lists the speed and throughput performance for different customized hardware running a kernel of size $h = 64, w = 64, k = 256, c = 256$. In all experiments, we round the dynamically-generated offsets to integers. We use $8 \times 8 \times 9$ Multiply-Accumulate (MAC) units in the 3×3 convolution engine for all full convolution experiments and 16×9 MACs for depthwise convolution experiments.

Baseline The baseline hardware implementation for the original 3×3 deformable convolution directly accesses the DRAM without going through any cache or buffering. In Figure 2, the baseline implementation directly accesses the input and output data through HP ports and ① DDR controller. The input addresses are first calculated from the offsets loaded from DRAM. The 3×3 *Deform M2S* engine then fetches and packs the inputs into parallel data streams to feed into the MAC units in the 3×3 *Conv* engine.

Caching One hardware optimization to leverage the temporal and spatial locality of the nonuniform input accesses is to add a cache to the accelerator system. As shown in Figure 4, we load the inputs from ② LLC through the ACP port in this implementation to reduce the memory access latency of the cached values. Since the inputs are sampled from offsets without specific patterns in the original deformable convolution, the cache provides adequate support to buffer inputs that might be reused in the near future. As shown in Table 2, adding LLC results in 26.7% and 13.2% reduction in latency for the original full and depthwise deformable convolution.

Buffering With the bounded range modification to the algorithm, we are able to use the on-chip memory to buffer all possible inputs. Similar to a line-buffer design for the original 3×3 convolution that stores two lines of inputs to exploit all input locality, we store $2N$ lines of inputs so that it is sufficient to buffer all possible inputs for reuse. This implementation includes the ③ Line Buffer in Figure 4. With the effective buffering strategy, we can see in Table 2 that the latency of a bounded deformable is reduced by 26.4% and 87.5% for full and depthwise convolution respectively in a system without LLC. In a system with LLC, the reduction is 2.1% and 80.9% respectively. The depthwise deformable convolution benefits more from adding the buffer as it is a more memory-bound operation. The compute-to-communication ratio for its input is *oc* times lower than the full convolution.

Parallel Ports The algorithm change to enforce a square-shape sampling pattern not only reduces the bandwidth requirements for loading the input indices in hardware, but also helps to improve the on-chip memory bandwidth. With a non-predictable memory access pattern to the on-chip memory, only one input can be loaded from the buffer at each cycle if all sampled inputs are store in the same line buffer. By constraining the shape of deformable convolution to a square with variable dilation, we are guaranteed to have three different line buffers with each storing three sampled points. We can thus have three parallel ports (④ Multi-ports in Figure 4) accessing different line buffers concurrently. This co-optimization improves the on-chip memory bandwidth and leads to another ~ 30% reduction in latency for depthwise deformable convolution.

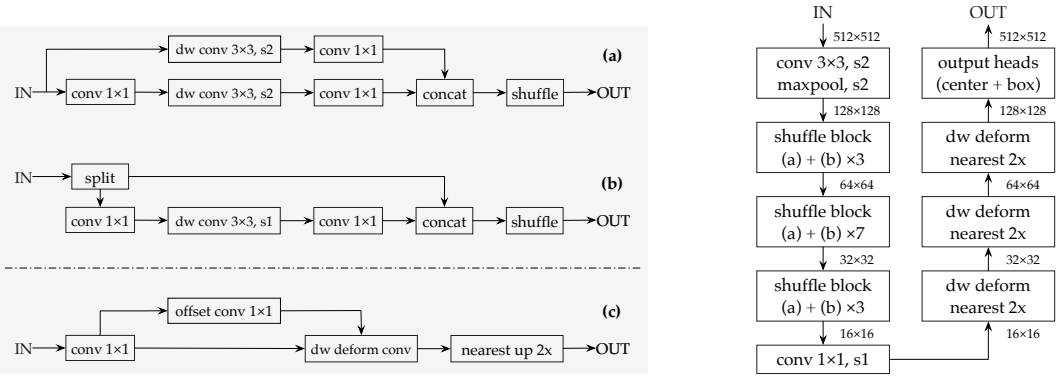
With the co-design methodology, our final result shows a 1.36× and 9.76× speedup respectively for the full and depthwise deformable convolution on the embedded FPGA accelerator.

4 DETECTION SYSTEM CO-DESIGN

In addition to the deformable convolution operation, the design of feature extractor, detection heads, and quantization strategy, also significantly impact the accuracy and efficiency of our detection system. In this section, we introduce CoDeNet for efficient object detection as well as a specialized FPGA accelerator design to support CoDeNet.

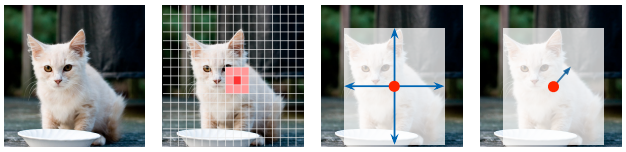
4.1 CoDeNet Design

To exploit the full potential of hardware acceleration, we carefully select and integrate the operations and building blocks in CoDeNet.



(i) building blocks (ii) model architecture

Figure 5: The architecture diagrams of our building blocks and model architecture.



(a) image (b) center heatmap (c) width & height (d) local shift

Figure 6: The output heads of CenterNet for object detection.

We devise CoDeNet to have the following embedded hardware compatible properties compared to other off-the-shelf network designs: 1) more uniform operation types to reduce the control complexity in the accelerator and to increase the accelerator utilization, 2) less computation to lower the overall latency to run on the embedded accelerator with limited compute capability, 3) smaller weights and inputs to be buffered on-chip for maximal reuse on the accelerator. Figure 5 shows the basic building blocks as well as the overall network architecture of CoDeNet.

Building Blocks and Feature Extractor The shaded part of Figure 5 shows the basic building blocks of CoDeNet. Building block (a) is used to down-sample the input images. A 3×3 depthwise convolution block with stride 2 is added to both of its branches together with 1×1 convolution to aggregate information across the channel dimension. Building block (b) splits the input features into two streams across the channel dimension. One branch is directly fed to the concatenation. The other streams through a sub-block of 1×1, 3×3 depthwise, and 1×1 convolution. This technique is referred to as identity mapping [12], which is commonly used to address the vanishing gradient problem during deep neural network training. Building block (a) and (b) together form a shuffle block as shown in the left branch of the overall architecture in Figure 5 as part of the feature extractor ShuffleNetV2 design. We choose ShuffleNetV2 as it is one of the state-of-the-art efficient network design. ShuffleNetV2 1x configuration only requires 2.3M parameters (4.8× smaller than ResNet-18 [11]) and 146M FLOPs of compute with resolution 224 × 224 (12.3x smaller than ResNet-18). Its top-1 accuracy is 69.4% on ImageNet (0.36% lower than ResNet-18).

The deformable operation is used in building block (c). Building block (c) is used to upsample the backbone features. The first 1×1 convolution is designed to map input channels to output channels. The following 3×3 depthwise deformable convolution samples the previous feature map, according to the offsets generated by 1×1 convolution. After that, a 2× upsampling layer, operated by a nearest neighbor kernel, is utilized to interpolate the higher resolution features. Note that, aside from the first layer, we only use 1×1 convolution and 3×3 depthwise (deformable) convolution in our build blocks. This way the building blocks of the whole network become more uniform and simple to support with specialized hardware.

Detection Heads As mentioned in Section 2.1, we use the anchor-free CenterNet [41] method to directly predict a gaussian distribution for object keypoints over the 2D space for object detection. Given an image $I \in \mathbb{R}^{W \times H \times 3}$, our feature extractor generates the final feature map $F \in \mathbb{R}^{\frac{W}{R} \times \frac{H}{R} \times D}$, where R is the output stride and D is the feature dimension. We set $R = 4$ and $D = 64$ for all the experiments. As illustrated in Figure 6, the outputs include:

(1) the keypoint heatmap $\hat{Y} \in [0,1]^{\frac{W}{R} \times \frac{H}{R} \times C}$
(2) the object size $\hat{S} \in \mathbb{R}^{\frac{W}{R} \times \frac{H}{R} \times 2}$
(3) the local offset $\hat{O} \in \mathbb{R}^{\frac{W}{R} \times \frac{H}{R} \times 2}$

Here C is pre-defined as 20 and 80 for VOC and COCO, respectively. In order to reduce the computation, we follow the class-agnostic practice, using the single size and offset predictions for all categories. To construct bounding boxes from the keypoint prediction, we first collect the peaks in keypoint heatmap $\hat{Y}$ for each category independently. Then we only keep the top 100 responses which are greater than its eight-connected neighborhood. Specifically, we use the keypoint values $\hat{Y}_{x_i y_i c}$ as the confidence measure of the i-th object for category c. The corresponding bounding box is decoded as $(\hat{x}_i + \delta\hat{x}_i - \hat{w}_i/2,\ \hat{y}_i + \delta\hat{y}_i - \hat{h}_i/2, \hat{x}_i + \delta\hat{x}_i + \hat{w}_i/2,\ \hat{y}_i + \delta\hat{y}_i + \hat{h}_i/2)$, where $(\delta\hat{x}_i, \delta\hat{y}_i) = \hat{O}_{\hat{x}_i \hat{y}_i}$ is the offset prediction and $(\hat{w}_i, \hat{h}_i) = \hat{S}_{\hat{x}_i \hat{y}_i}$ is the size prediction.

Quantization Quantization is a crucial step towards the efficient deployment of the GPU pre-trained model on FPGA accelerators. Although many previous works treat quantization as a separate process outside the algorithm-hardware co-design loop, we note that quantization performance greatly depends on the network architecture. As an example, the residual connection will enlarge the activation range of specific layers, which makes a uniform quantization setting sub-optimal. And it requires a special design

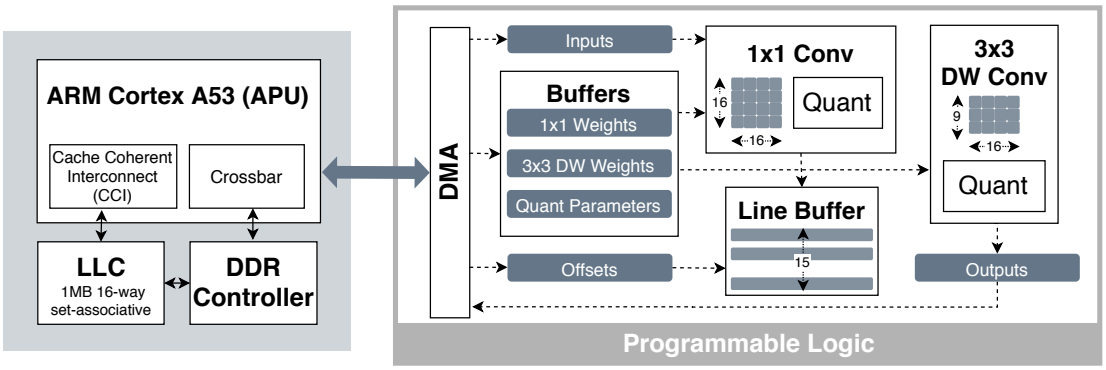


Figure 7: Architectural diagram of the FPGA accelerator.

for addition in int32 format, otherwise, extra steps of quantization are needed to support the low-precision addition. With this prior knowledge, we use concatenation instead of residual connection throughout CoDeNet, and we do not use techniques such as layer aggregation [35], in order to achieve a simpler hardware design. We adopt a symmetric uniform quantizer shown as follows:

$$X' = \text{clamp}(X, -t, t), \tag{1}$$

$$X^I = \lfloor \frac{X'}{\Delta} \rceil, \text{ where } \Delta = \frac{t}{2^{k-1}-1}, \tag{2}$$

$$Q(X) = \Delta X^I, \tag{3}$$

where Q stands for quantization operator, X is a floating point input tensor (activations or weights), $\lfloor \cdot \rceil$ is the round operator, Δ is the quantization step (the distance between adjacent quantized points), X^I is the integer representation of X, and k is the quantization precision for a specific layer. Here, threshold value t determines the quantization range of the floating point tensor, and the clamp function sets all elements smaller than $-t$ to $-t$, and elements larger than t to t. It should be noted that the threshold value t can be smaller than max or $|min|$ in order to get rid of outliers and better represent the majority of a specific tensor. In order to achieve better AP, we perform 4-bit channel-wise quantization [14] for weights. Meanwhile, to ease the hardware design and accelerate the inference, we choose symmetric uniform quantizer rather than non-uniform quantizer, and we use 8-bit layer-wise quantization for activations. During quantization-aware fine-tuning, we use Straight-Through Estimator (STE) [1] to achieve the backpropagation of gradients through the discrete operation of quantization.

4.2 Dataflow Accelerator

We develop a specialized accelerator to support the aforementioned CoDeNet design on an FPGA SoC. As shown in Figure 7, the FPGA SoC includes the programmable logic (PL), memory interfaces, a quad-core ARM Cortex-A53 application processor with 1MB LLC, and etc. Our accelerator in the PL side communicates to the processor through an AXI system bus. The High Performance (HP) and Accelerator Coherency Port (ACP) interfaces on the AXI bus allow the accelerator to directly access the DRAM or perform cache-coherent accesses to the LLC and DRAM. The processor provides software support to invoke the accelerator and to run functions that are not implemented on the accelerator.

With our co-design methodology, we are able to reduce the types of operations to support in the accelerator. Excluding the first layer for the full 3×3 convolution, CoDeNet only consists of the following operations: (i) 1×1 convolution, (ii) 3×3 depthwise (deformable) convolution, (iii) quantization, (iv) split, shuffle and concatenation. This helps us simplify the complexity of the control logic and thus saves more FPGA resources for the actual computation. We partition the CoDeNet workload so that the frequently-called compute-intensive operations are offloaded to the FPGA accelerator while the other operations are run by software on the processor. The operations we choose to accelerate are 1×1 convolution, 3×3 depthwise (deformable) convolution, and quantization, with the other operations offloaded to the processor.

To leverage both the data-level and the task-level parallelism, we devise a spatial dataflow accelerator engine to execute a subgraph of the CoDeNet at a time and store the intermediate outputs to the DRAM. In the dataflow engine, the execution of compute units is determined by the arrival of the data and thus further reduces the overhead from the control logic. As illustrated in the architectural diagram in Figure 7, our accelerator executes 1×1 convolution with quantization and 3×3 depthwise (deformable) convolution with quantization in order. We implement the accelerator with Vivado HLS and its dataflow template. All functional engines are connected to each other through data FIFOs. Extra bypass signals can be asserted if the user would like to bypass either of the main computation blocks. By co-designing the network to use operations with fewer weight parameters, such as depthwise convolution, we are able to buffer the weights for all operations in the on-chip memory and enable the maximal reuse of the weights once they are on-chip. We also add a line buffer for the 3×3 depthwise (deformable) convolution to maximize the reuse of inputs on-chip. This optimization is enabled by the operation co-design discussed in Section 3.2. The line buffer stores 15 rows of the input image. The size of this buffer is larger than $15 \times w \times ic$ of any layers in the CoDeNet design. Our input tensors are laid out in the NHWC manner, allowing the data along the channel dimension C to be stored in contiguous memory blocks.

1×1 **convolution** The compute engine for the 1×1 convolution is composed of 16×16 multiply-accumulate (MAC) units. At each round of the run, the engine takes 16 inputs along its channel dimension and broadcasts each of them to 16 MAC units. Meanwhile,

Detector	Weights	Activations	Model Size	MACs	AP50
Tiny-YOLO	32-bit	32-bit	60.5 MB	3.49 G	57.1
CoDeNet 1×	32-bit	32-bit	6.06 MB	1.14 G	64.6
CoDeNet 1×	4-bit	8-bit	0.76 MB	1.14 G	61.7
CoDeNet 2×	32-bit	32-bit	23.2 MB	3.58 G	69.6
CoDeNet 2×	4-bit	8-bit	2.90 MB	3.58 G	67.1

Table 3: Quantized CoDeNet on VOC object detection.

Detector	Weights	Model Size	MACs	AP	AP50	AP75	APs	APm	APl
CoDeNet 1×	32-bit	6.07MB	1.24G	21.1	36.5	21.1	4.1	21.7	36.3
CoDeNet 1×	4-bit	0.76MB	1.24G	17.4	31.9	17.4	3.5	17.1	30.5
CoDeNet 2×	32-bit	23.4MB	4.41G	26.1	43.3	26.8	7.0	27.9	43.5
CoDeNet 2×	4-bit	2.93MB	4.41G	20.6	36.4	20.6	5.6	22.3	35.2

Table 4: Quantized CoDeNet on COCO object detection.

it unicasts 16×16 weights for 16 inputs channels and 16 output channels to their corresponding MAC unit. There are 16 reduction trees of size 16 connected with the MAC units to generate 16 partial sums of the products. The partial sums are stored on the output registers and are accumulated across each round of the run. Every time the engine finishes the reduction along the input channel dimension, it feeds the values of the output registers to the output FIFO and resets their values to zero.

3×3 **depthwise (deformable) convolution** This engine directly reads 16 sampled 3×3 inputs from the line buffer design and multiplies them by 3×3 weights from 16 corresponding channels. Then it computes the outputs with 16 reduction trees to accumulate the partial sums of along 3×3 spatial dimension. Both the original and the deformable depthwise convolutions can be run on this engine. The original depthwise operation is realized by hardcoding the offset displacement to be 1.

Quantization To convert the output from 16-bit sum to 8-bit inputs, we add a quantization unit at the end of each compute engine. The quantization unit multiplies each output with a scale, and then add a bias to it. It returns the lower 8 bits of the result as the quantized value. The parameters, such as the scale and bias for each channel, are preloaded to on-chip buffer to save the memory access time. Note that we also merge the batch normalization and ReLU in this compute unit. We follow the practice introduced in [13] to perform integer inference for our quantized model.

5 EXPERIMENTAL RESULTS

We implement CoDeNet in Pytorch, train it with a pretrained ShuffleNetV2 model, and quantize the network to use 8-bit activations and 4-bit weights. We devise several configurations of CoDeNet to facilitate the latency-accuracy tradeoffs for our final object detection solution on the embedded FPGAs. Different configurations of the CoDeNet are listed in Table 3 and 4 showing the accuracies for object detection on Pascal VOC and COCO dataset. As shown in Table 3, compared to Tiny-YOLO, our compact 1× model is 10× smaller without quantization and 79.6× smaller with quantization, while achieving higher accuracy. In addition, the total MACs count of the compact design is 3.1× smaller than Tiny-YOLO. We also see that quantizing the model to 4-8 bits causes minor accuracy drop, but it significantly reduces the model size (> 8×).

We evaluate our accelerator tailored for CoDeNet on the Ultra96 development board with Xilinx Zynq XCZU3EG UltraScale+ MPSoC

LUT	FF	BRAM	DSP
34144 (48.4%)	41827 (29.6%)	216 (100%)	360 (100%)

Table 5: FPGA resource utilization.

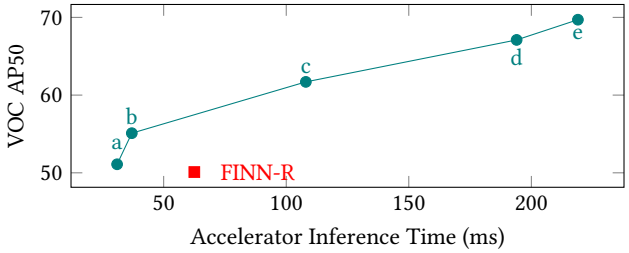


Figure 8: Latency-accuracy trade-off on VOC.

	Platform	Framerate (fps)	Test Dataset	Precision	Accuracy
DNN1 [10]	Pynq-Z1	17.4	DJI-UAV	a8	IoU(68.8)
DNN3	Pynq-Z1	29.7		a16	IoU(59.3)
Skynet [38]	Ultra96	25.5		w11a9	IoU(71.6)
Finn-R [2] [26]	Ultra96	16	VOC07	w1a3	AP50(50.1)
Ours (config a)	Ultra96	32.2		w4a8	AP50(51.1)
Ours (config b)	Ultra96	26.9		w4a8	AP50(55.1)
Ours (config c)	Ultra96	9.3		w4a8	AP50(61.7)

Table 6: Performance comparison with prior works.

device. Our accelerator design runs at 250 MHz after synthesis, and place and route. Table 5 shows the overall resource utilization of our implementation. We observe a 100% utilization of both DSPs and BRAMs. Most DSPs are mapped to the 4-8 bit MAC units and BRAMs are mainly used for the line buffer design.

We provide a pareto curve in Figure 8 showing the latency-accuracy tradeoff for various CoDeNet design points with acceleration. Configuration *a* and *b* in this curve are trained and inferenced with images of size 256×256 instead of the original size 512×512. The smaller input image size leads to ~4× reduction in MACs. In configuration *a*, *c* and *d*, the stride of the first layer is increased to 4 from 2, which greatly reduces the first layer runtime on the processor. In configuration *d* and *e*, we use the CoDeNet 2× model, where the channel size is doubled in the network, to boost the accuracy. A comparison of our solutions against previous works is shown in Table 6. We found that very few prior works on embedded FPGAs attempt to target the standard dataset like VOC or COCO for object detection primarily due to the limit of hardware resources and inefficient model design. Two state-of-the-art FPGA solutions that meet the real-time requirement in the DAC-UAV competition target the DJI-UAV dataset for drone image detection. However, object detection on DJI-UAV is less generic and less difficult than on VOC or COCO since the images in DJI-UAV are from the top-down view with few overlapped objects. Therefore, we do not target DJI-UAV in this work and aim to provide a more general solution. Compared to the results from FINN-R, the state-of-the-art work targeting VOC, our configuration *a* and *b* achieve both higher accuracy and higher framerate. The latency evaluation on our accelerator is done with batch size equal to 1 without any runtime parallelization. In configuration *a* and *b*, all layers excluding the first one takes 29.9 ms to finish on the accelerator, the latency of the first layer is 1.2ms and 3.1ms respectively on the processor. Results in Figure 8 show that CoDeNet designs are on the pareto-optimal front of the latency-accuracy tradeoff targeting VOC.

6 CONCLUSION

In this work, we perform a detailed accuracy-efficiency trade-off study for each hardware-friendly algorithmic modification to the deformable convolution operation, with the goal of co-designing an efficient object detection network and a real-time embedded accelerator optimizing for accuracy and speed. Results show that these modifications lead to significant hardware performance improvement in the accelerator with minor accuracy loss. Our co-designed model CoDeNet with the modified deformable convolution is 79.6× smaller than Tiny YOLO and its corresponding embedded FPGA accelerator is able to achieve realtime processing with a framerate of 26.9.

REFERENCES

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[2] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[3] Yuntao Chen, Chenxia Han, Yanghao Li, Zehao Huang, Yi Jiang, Naiyan Wang, and Zhaoxiang Zhang. 2019. Simpledet: A simple and versatile distributed framework for object detection and instance recognition. *JMLR* (2019).

[4] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *CVPR*.

[5] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. 2017. Deformable convolutional networks. In *ICCV*.

[6] Zhen Dong, Zhewei Yao, Yaohui Cai, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks. *arXiv preprint arXiv:1911.03852* (2019).

[7] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *ICCV*.

[8] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. Centernet: Keypoint triplets for object detection. In *ICCV*.

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *IJCV* (2010).

[10] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge. *arXiv preprint arXiv:1904.04421* (2019).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *ECCV*.

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*.

[14] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[15] Hei Law and Jia Deng. 2018. Cornernet: Detecting objects as paired keypoints. In *ECCV*.

[16] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. 2019. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900* (2019).

[17] Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. 2019. Fully quantized network for object detection. In *CVPR*.

[18] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. 2019. Scale-aware trident networks for object detection. In *ICCV*.

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *ICCV*.

[20] 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 *ECCV*.

[21] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. 2018. Path aggregation network for instance segmentation. In *CVPR*.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *ECCV*.

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*.

[24] Yufei Ma, Tu Zheng, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2018. Algorithm-hardware co-design of single shot detector for fast object detection on FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[25] 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*. ACM, 31–40.

[26] 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.

[27] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *CVPR*.

[28] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *CVPR*.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*.

[31] Evan Shelhamer, Dequan Wang, and Trevor Darrell. 2019. Blurring the line between structure and learning to optimize and adapt receptive fields. *arXiv preprint arXiv:1904.11487* (2019).

[32] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Q-bert: Hessian based ultra low precision quantization of bert. In *AAAI*.

[33] Ke Xu, Xiaoyun Wang, and Dong Wang. 2019. A Scalable OpenCL-Based FPGA Accelerator for YOLOv2. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 317–317.

[34] Xiaowei Xu, Xinyi Zhang, Bei Yu, X Sharon Hu, Christopher Rowen, Jingtong Hu, and Yiyu Shi. 2019. Dac-sdc low power object detection challenge for uav applications. *TPAMI* (2019).

[35] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. 2018. Deep layer aggregation. In *CVPR*.

[36] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*.

[37] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. 2020. ResNeSt: Split-Attention Networks. *arXiv preprint arXiv:2004.08955* (2020).

[38] Xiaofan Zhang, Yuhong Li, Cong Hao, Kyle Rupnow, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2019. SkyNet: A Champion Model for DAC-SDC on Low Power Object Detection. *arXiv preprint arXiv:1906.10327* (2019).

[39] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).

[40] 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).

[41] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. 2019. Objects as points. *arXiv preprint arXiv:1904.07850* (2019).

[42] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krahenbuhl. 2019. Bottom-up object detection by grouping extreme and center points. In *CVPR*.

[43] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. 2019. Deformable convnets v2: More deformable, better results. In *CVPR*.