Light-Weight RetinaNet for Object Detection on Edge Devices

Yixing Li
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Email: yixingli@asu.edu

Akshay Dua
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Email: adua5@asu.edu

Fengbo Ren
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Email: renfengbo@asu.edu

***Abstract*—This paper aims at reducing computation for Retinanet, an mAP-30-tier network, to facilitate its practical deployment on edge devices for providing IoT-based object detection services. We first validate RetinaNet has the best FLOP-mAP trade-off among all mAP-30-tier network. Then, we propose a light-weight RetinaNet structure with effective computation-accuracy trade-off by only reducing FLOPs in computationally intensive layers. Compared with the most common way of trading off computation with accuracy–input image scaling, the proposed solution shows a consistently better FLOPs-mAP trade-off curve. Light-weight RetinaNet achieves a 0.3% mAP improvement at 1.8x FLOPs reduction point over the original RetinaNet, and gains 1.8x more energy-efficiency on an Intel Arria 10 FPGA accelerator in the context of edge computing. The proposed method potentially can help a wide range of the object detection applications to move closer to a preferred corner for a better runtime and accuracy, while enjoys more energy-efficient inference at the edge.**

I. Introduction

The Internet-of-Things (IoT) boosts the vast amount of streaming data. Due to the limited bandwidth and the latency of cloud computing, it calls for the need of offloading the computing to decentralized edge computing infrastructures [1]. On the other side, deep learning-based applications take the advantage of heavy computing resources for training large models to fit more and more complicated tasks. Even though the model performs well in terms of accuracy, its complexity may even make it impossible to be deployed on the edge devices. In order to enable more IoT services on edge devices, it is essential to study the effective computation and accuracy trade-off of the deep learning-based models and tailor it for IoT services on edge devices.

Object detection is the key module in face detection, tracking objects, video surveillance, pedestrian detection, etc. [17], [18]. With the recent development of deep learning, it boosts the performance of object detection tasks. However, regarding the computational complexity (in terms of FLOPs), a detection network can possibly consume three orders of magnitude more FLOPs than a classification network, which makes it much more challenging to be deployed on an edge device.

With detection on edge devices, convenience stores with gas stations can do real-time analysis of customers' age and gender for personalized on-screen recommendation [16]. Comparing to detection in self-driving cars, a hard real-time scenario (30 fps), the in-store real-time recommendation system only needs a soft real-time response (1-2 fps) and also has a certain tolerance of the accuracy rate. To hit this target frame rate with edge devices, the reasonable object detection solutions fall in the upper mid range – mAP-30-tier.

In this paper, we use the mAP as the indicator to categorize the existing object detection solutions. The mAP-20-tier solutions are the most aggressive ones that target highly energy- and resource-constrained devices, such as battery-powered mobile devices. The existing solutions, such as YOLOv1, YOLOv2, SSD, MobileNetv2-SSDLite [17], [18], [14] have pushed hard to reduce the memory consumption by trading off their accuracy performance. Their detection accuracy on the large dataset (COCO test-dev 2017 [13]) yields to mAP of 22-25%. The mAP-40-tier solutions, such as MaskRCNN and its variations, on the contrary, are targeting the best mAP performance with less concern about computation resources. The mAP-30-tier solutions do not sacrifice the accuracy performance too much but are more aware of the computational efficiency. In the mAP-30-tier, popular solutions include Faster R-CNN, RetinaNet, YOLOv3 [20], [12], [19], and their variants. These solutions can be potentially deployed on edge GPUs (e.g., Nvidia T4 GPU) or FPGAs (e.g., Intel Arria 10 FPGA based acceleration card), since the on-board memory resources are generally enough for preloading the weights of the mAP-30-tier networks. [10] verifies the linear relation between FLOP count and inference runtime for the same kind of network.

The overview of the FLOP-mAP profile of the existing mAP-20-tier and mAP-30-tier detection networks are shown in Fig. 1. The upper-left corner is the preferred corner in the FLOP-mAP plane. We take the RetinaNet as the baseline since it has the best FLOP-mAP trade-off in the mAP-30-tier. We use the FLOP count as a key indicator for comparison. By applying Faster R-CNN [20], RetinaNet [12] and YOLOv3 [19] on the same task for COCO detection dataset, which takes an input image around 600×600-800×800, the mAP will hit in the range of 33%-36%. However, the FLOPs of Faster R-CNN [20] is around 850 GFLOPs (gigaFLOPs), which is at least 5x more than that of RetinaNet and YOLOv3 [19]. Apparently, Faster R-CNN is not competitive in terms of computational efficiency. From YOLOv2 [18] to YOLOv3 [19], it is inter-

978-1-7281-5503-6/20/$31.00 ©2020 IEEE

esting that the authors have aggressively increased the number of FLOPs from 30 to 140 GFLOPs to gain mAP improvement from 21% to 33%. Even with that, the mAP of YOLOv3 is 2.5% lower than RetinaNet with 150 GFLOPs. Also, a low-end version of MaskRCNN [7] with mAP of 37.8% cannot beat RetinaNet in terms of runtime. These observations inspire us to take the RetinaNet as the baseline to explore the feasibility of creating a light-weight version of it.

There are two common methods to reduce the FLOPs in a detection network. One way is to switch to another backbone, while the other is to reduce the input image size. The first method results in noticeable accuracy drop if one substitutes a ResNet backbone [8] with a more shallow one. Typically it is not considered as a good accuracy-FLOP trade-off scheme with a small variation. With regard to reducing the input image size, it is an intuitive way to reduce the FLOPs. However, the accuracy-FLOP trade-off curve shows degradation in a polynomial trend [10]. There is an opportunity to find a more linear degradation tendency curve for a better accuracy-FLOP trade-off. We propose only to replace certain branches/layers of the detection network with light-weight architecture and keep the rest of the network unchanged. For the RetinaNet, the heaviest branch is the succeeding layers of the finest FPN (P3 in Fig. 2), which takes up to 48% of the total FLOPs. We propose different light-weight architecture variants. Moreover, the proposed method can also be applied to other blockwise-FLOPs-imbalance detection networks.

The contribution of this paper is three-fold. First, we analyze the accuracy-computation trade-off of RetinaNet and propose a light-weight RetinaNet model structure by simplifying the heaviest bottleneck layer. Second, we illustrate that the proposed light-weight RetinaNet has a constantly better FLOP-mAP trade-off curve (linear degradation) than a naive input image scaling approach (polynomial degradation). Third, we quantitatively evaluate the runtime performance on an FPGA-based edge node and show that the proposed method results in 0.3% mAP improvement at 1.8x FLOP reduction with no sacrifice in runtime, compared with input image scaling method.

II. Related Work

The overview of the FLOP-mAP profile of the existing mAP-20-tier and mAP-30-tier detection networks are shown in Fig. 1. In the FLOP-mAP plane of Fig. 1, the upper-left corner is the preferred corner, which has the best FLOP-mAP tradeoff.

A. mAP-30-tier object detection networks

Faster RCNN [20] is an advanced architecture, which boosts both the accuracy and runtime performance from R-CNN and Fast R-CNN [4], [3]. As Faster RCNN [20] replaces the selective search (used by Fast R-CNN) with RPN, it significantly reduces the runtime of generating the region proposals. However, in the inference stage of Faster R-CNN, there are still around 256-1000 boxes feeding into the detection network. It is really expensive to process this much data fed in by the proposed boxes. As for the Faster RCNN to process COCO dataset detection task with an Inception-ResNetV2 [21] backbone, the total numbers of FLOPs can come up to over 800 GFLOPs.

Compared with Faster RCNN [20], the RetinaNet [12] targets a simpler design for gaining speedup. A feature pyramid network (FPN) [11] is attached to its backbone to generate multi-scale pyramid features. Then, pyramid features go into classification and regression branches, whose weights can be shared across different levels of the FPN. The focal loss is applied to compensate for the accuracy drop, which makes its accuracy performance to be comparable with the Faster RCNN.

B. mAP-20-tier object detection networks

The YOLO network family is among the most popular ones in the mAP-20-tier. The most distinguishing feature of YOLO is its predefined grid cell. The input image can be cut into S×S grid cells, and each cell only predicts one object. This idea apparently helps to reduce the computation complexity. However, in the meantime, it increases the chances of undetected objects and has relatively bad performance in detecting small objects. The YOLOv1 [17] is only evaluated on relatively small datasets (PASCAL VOC), aiming at enabling real-time inference. From YOLOv2 [18] to YOLOv3 [19], the mAP performance results on COCO test-dev2015 dataset is boosted from 21.6% to 33.0%. It's worth noting that the accuracy gain of YOLOv3 comes along with the FLOPs increment from 63 GFLOPs to 141 GFLOPs. YOLOv3 [19] should not be categorized as a light-weight one anymore, as the FLOP count and mAP are closed to those of RetinaNet-ResNet50-FPN (156 GFLOPs and mAP = 35.7%). We also include other light-weight object detection networks such as SSD and SSDLite [14] in Fig.1 for providing an overview of the FLOP-mAP profile of mAP-20-tier detection networks.

C. Others and discussions

For the most accurate detection networks, the mAP-40-tier, the representative work is the MaskRCNN [7]. In the benchmark study of model zoo [5], at the mAP = 37.8%, MaskRCNN is less computation-efficient than RetinaNet. Also, it mainly targets at high detection accuracy with less constraint on computational complexity. While the mAP-20-tier ones are extremely compact, aiming at highly energy- and resource-constrained devices, such as the battery-powered mobile devices. The mAP-30-tier ones are more suitable for edge deployment, and the RetinaNet can win over any others in FLOP-mAP tradeoff. These inspires us to take the RetinaNet as the baseline design to explore a better scheme for accuracy and FLOPs trade-off for the mAP30-tier detection on edge.

III. Light-weight RetinaNet

In this section, we first analyze the RetinaNet network with a focus on the distribution of the number of floating-point operations (FLOPs) across different layers in Section III.A. Then, we discuss the approach for creating light-weight RetinaNet in Section III.B.

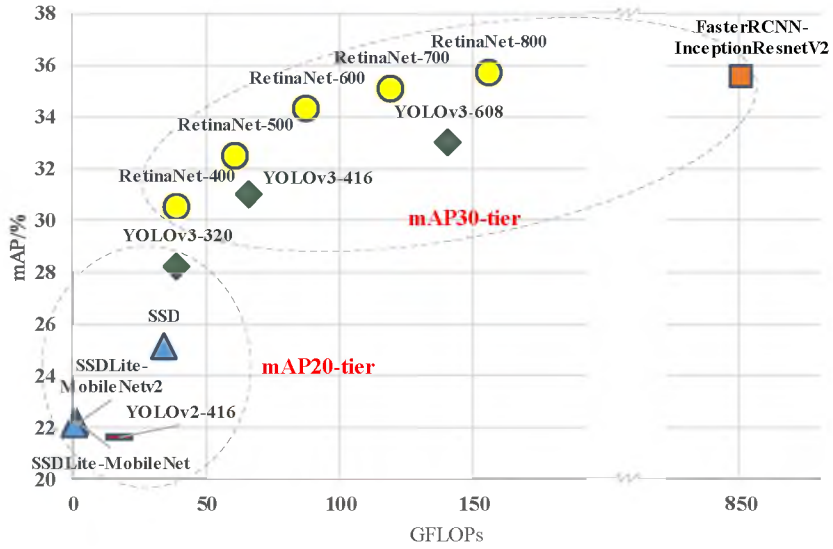

Fig. 1. An overview of the mAP-20-tier and mAP-30-tier detection networks.

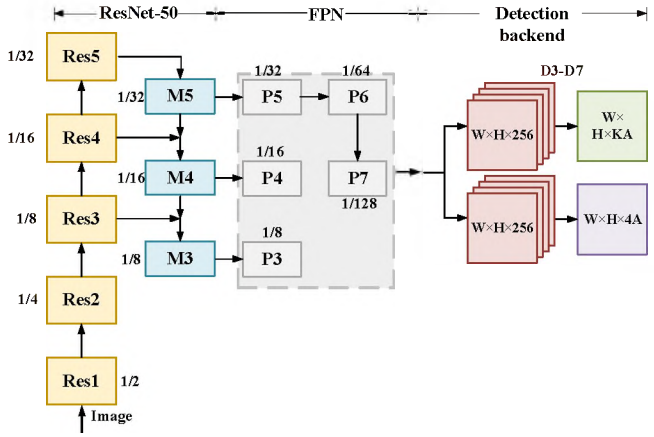

Fig. 2. RetinaNet (ResNet50-FPN-800x800) network architecture.

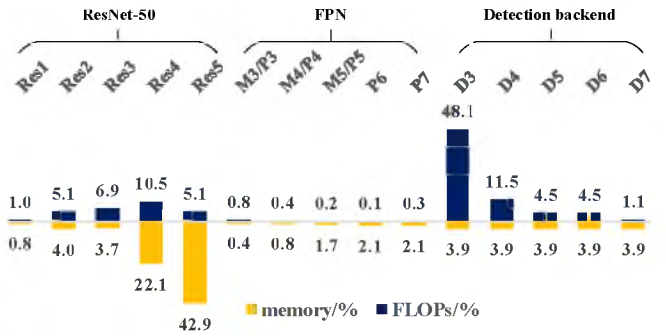

Fig. 3. The FLOPs and memory (parameter) distribution of RetinaNet (ResNet50-FPN-800x800) across different blocks.

A. *RetinaNet Primer*

The RetinaNet architecture is composed of three parts – a backbone, a feature pyramid network (FPN) [11], and a detection backend, as shown in Fig. 2. The image is first processed by the backbone, which usually is the ResNet Architecture. Here, it is worthy of noting that although MobileNet's performance [9] is on a par with ResNet in classification tasks, MobileNet may not be a good alternative to ResNet for detection tasks. From both [10] and our observation, using MobileNet [9] as the backbone for detection tasks will suffer from much more accuracy drop than it does for classification tasks. The main reason is that the confidence scores of a MobileNet-based backbone are the trade-off for lower computation costs. Therefore, a MobileNet-based backbone is hardly a desirable choice for high precision object detection networks. The backbone, together with the subsequent FPN forms an encoder-decoder-like network. The benefit of the FPN is that it merges the features of consecutive layers from the coarsest to the finest level. After that, the multi-scale pyramid features (P3-P7) feed into the backend where two detection branches are used for bounding box regression and object classification. Note that the detection branch and bounding box branch do not share weights. The weights of each branch are shared across the pyramid features (P3-P7).

The FLOP distribution of RetinaNet architecture (ResNet50-FPN-800x800, as shown in Fig. 2) across different blocks is shown in Fig. 3., where each block corresponds to the same block in Fig. 2. The detection backend D3-D7 is the succeeding layer of P3-P7, respectively. As in the original design, D3-D7 share the same weight parameters, the average memory cost of D3-D7 is shown in Fig. 3. The FLOP count of the D3 block dominates the total FLOP count at 48.1%. This unbalanced FLOP distribution is quite different from that of the ResNet architecture, which has a small FLOP count variance across different blocks. The unbalanced FLOP distribution presents an opportunity to get a meaningful overall FLOP reduction at little cost of accuracy drop by only reducing computational complexity of the heaviest layer. Specifically, if we can reduce the FLOPs of D3 by half, the total FLOPs can be reduced by 24%.

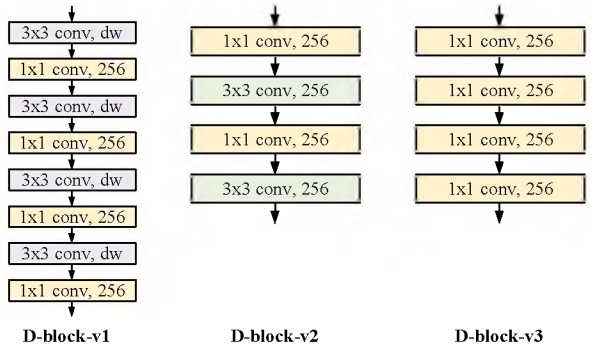

Fig. 4. light-weight blocks for detection backend.

B. Tiny backend solution

1) Light-weight block: Intuitively, we can reduce the filter size to get FLOP reduction. As shown in Fig. 4, we propose different block designs for the detection branches of ResNet. The D-block-v1 applies the MobileNet [9] building block. A 3×3 depth-wise (dw) convolution is followed by a 1×1 convolutional block to substitute an original layer. The D-block-v2 alternately uses the 1×1 and 3×3 kernel. It is inspired by YOLOv1 [17], which has replaced the 3×3 kernels without introducing residual blocks. The reduction of D-block-v3 is more aggressive, which replaces all the 3×3 convolutions with 1×1 convolutions. In general, if one substitutes a given block with a more light-weight block, it will cause accuracy drop of the network as a trade-off for less computation cost. In our case, we also observe accuracy (mAP) drop if we replace the original blocks with light-weight blocks in detection backend. Therefore, we propose to add limited overheads to compensate for the accuracy drop here with a partially shared weights scheme.

2) Partially shared weights: As illustrated in Section III.B.1, the light-weight detection blocks is to trade off lower computational complexity with accuracy drop. To compensate for the accuracy drop, we propose to replace the fully shared weight scheme in the original RetinaNet with a partial shared weight scheme. As shown in Fig. 2, P3-P7 are the multi-scale feature map outputs of FPN, which are then fed into detection backend D3-D7, respectively. Although D3-D7 share the weight parameters, D3-D7 have different input sizes (P3-P7), respectively, and D3-D7 are processed in serial. Fig. 5(a) is the original detection backend that D3-D7 fully share the weights. In Fig. 5(b), only D4-D7 share the weights with the original configuration, while D3 is processed by the light-weight D-block-v1/v2/v3 proposed in Section III.B.1.

The proposed partially shared weights scheme mainly has two advantages. First, as D3 has its independent weight parameters, it can learn more tailored features at its feature map scale (D3-D7 have different sizes of feature maps), which can compensate for the accuracy drop brought by reducing computational complexity. Second, it allows us to focus on reducing the computational complexity of the heaviest bottleneck block without touching the rest of the network. By doing so, D4-D7 are produced by exact the same architecture as the D4-D7 in the original RetinaNet, which should guarantee

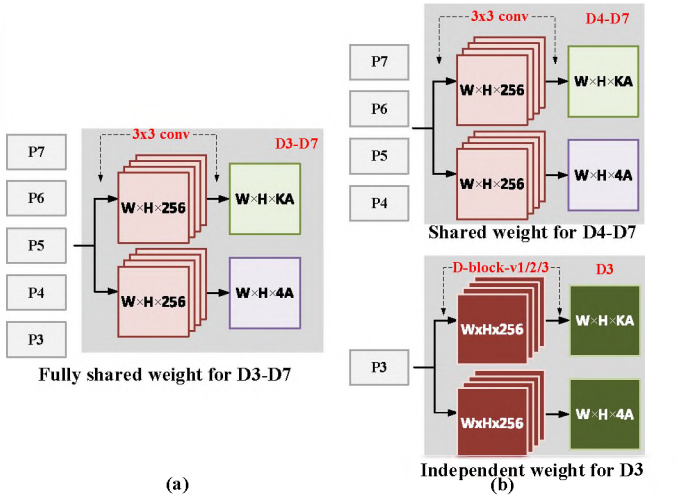

Fig. 5. Fully and partially shared weights for detection backend.

the same performance for D4-D7 outputs. Also, since the backbone (ResNet-50) dominates the memory consumption (as shown in Fig. 3), the overhead of memory consumption here (less than 1%) can be negligible.

IV. Results and Discussion

A. Experimental setup

To measure the accuracy performance, we perform our experiments in Caffe2 with 4 Titan X GPUs. We build upon the open-source code of RetinaNet in [5]. As the original work is trained with 8 GPUs, we scale down the base learning rate by 2x and extend the training epochs by 2x, as suggested in [6]. Besides, since [15] proves the deep neural network is less easy to overfit when its computational complexity is reduced by network compression, we further extend the training epochs (by the same ratio of FLOP reduction) for getting a better accuracy rate. In all the experiments, we use the same the network configuration as RetinaNet-ResNet50-FPN. The source code is available online at https://github.com/PSCLab-ASU/LW-RetinaNet/.

To evaluate the runtime performance on FPGA-based edge devices, we mapped RetinaNet and light-weight RetinaNet on an Intel Arria 10 GX 1150 FPGA acceleration card hosted by an Linux edge server. Intel FPGA SDK for OpenCL version 18.0 is used to compile the device code. The host code is written in C/C++ and the device code in OpenCL C language. SystolicArrayCNN – an open-source optimized OpenCL kernel is used for CNN acceleration [2][1]. Each layer is run with an optimized OpenCL-based FPGA kernel for the runtime and power evaluation.

B. Performance on COCO dataset

The COCO dataset [13] is considered as the most challenging dataset for object detection. We only perform experiments on COCO dataset (so does the original RetinaNet [12]). We train the light-weight RetinaNet on the 2017 COCO training dataset and test it on the COCO test-dev.

[1]https://github.com/PSCLab-ASU/SystolicArrayCNN

TABLE I
COMPARISON BETWEEN DIFFERENT LIGHT-WEIGHT BLOCK.

Light-weight block	scale	mAP	ΔmAP\%	GFLOPs	ΔFLOPs/%
original	800	35.7	0	156	0
D-block-v1	800	34.3	1.4	135	15.4
D-block-v2	800	35.6	0.1	135	6.4
D-block-v3	800	35.1	0.6	89	15.4

TABLE II
CONFIGURATIONS OF DIFFERENT LIGHT-WEIGHT(LW) RETINANET.

	Light-weight block	Detection backend	
		Classification	Bouding box
LW-RetinaNet-v1	D-block-v2	√	
LW-RetinaNet-v2	D-block-v3	√	
LW-RetinaNet-v3	D-block-v3	√	√

TABLE III
RESOURCE UTILIZATION OF INTEL ARRIA 10 GX 1150 FPGA IMPLEMENTATION.

Resource Type	Utilization amount	Percentage
Frequency	210 MHz	-
Logic utilization	248K / 427K	58%
DSP utilization	1,184 / 1,518	78%
BRAM utilization	1,818 / 2,713	67%

Table 1 shows the comparison among different light-weight blocks that we propose in Section III.B.1. In this set of experiments, we only use the light-weight block in the regression branch (for the bounding box) of detection backend, which is the upper branch shown in Fig. 2 detection backend. The results of Table 1 show that the D-block-v1 – the one with the MobileNet building block has 0.8% lower mAP compared with the D-block-v3, which has the same FLOP reduction percentile. It also aligns with our analysis in Section III.B that although MobileNet is proven to a powerful light-weight classification network architecture, MobileNet building block is not guaranteed to be the best building block substitution for other computer vision tasks. Therefore, with the same scale of FLOP reduction, we choose D-block-v3 over D-block-v1 in the following experiment. As the D-block-v2 performs less aggressive FLOP reduction, its mAP is only reduced by 0.1%, which is a good trade-off for a small scale FLOP reduction (15%).

The configurations for different versions of light-weight RetinaNet with D-block-v2 or D-block-v3 light-weight blocks are shown in Table 2. Specifically, Table 2 shows which light-weight block is applied to which branches of the backend in each version. The corresponding light-weight RetinaNet performance results are shown in Table 4. As scaling down input image size is the only method that proposed in existing work of FLOP-mAP trade-off for RetinaNet, we also cite the performance results of the original RetinaNet at different input scales from the original paper[12]. For better comparison between the proposed method and input image scaling method, we visualize the FLOPs and accuracy trade-off in Fig. 6. Each data point in Fig. 6 corresponds to one row of the results in

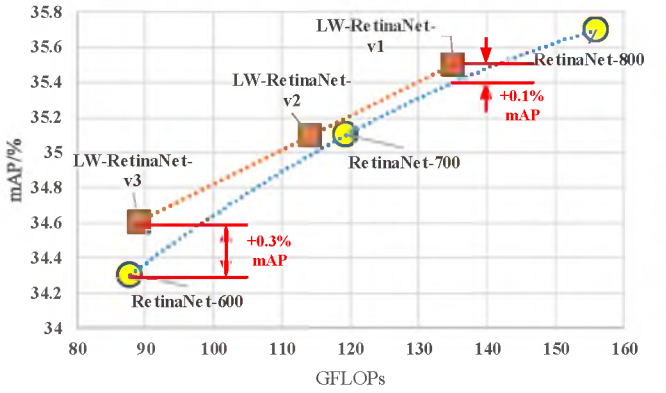

Fig. 6. FLOPs and mAP trade-off for input image size scaling versus the proposed method.

Table 4. We mark the trending curve of light-weight RetinaNet in red dot curve and that of original RetinaNet in blue dot curve. The upper-left corner is the preferred corner in the FLOP-mAP plane. As the red curve is constantly closer to the preferred corner, it indicates that the proposed method has a better FLOP-mAP trade-off than the conventional input image scaling method. The difference between these two methods results in a 0.1% mAP gap at the same number of FLOPs with low reduction ratio of 15%. However, as we further reduce the number of FLOPs, the proposed method shows a trend of linear degradation, while the input image scaling method degrade in a more polynomial fashion. Fig. 6 clearly shows a divergence around 90 GFLOPs, where the input scaling method yields to 0.3% more accuracy drop than the proposed method.

We use the experimental setup in IV.A to evaluate the runtime performance on FPGA-based edge devices. The resource utilization of the FPGA kernel mapped on an Intel Arria 10 GX 1150 FPGA board is shown in Table 3. The actual runtime shown in Table 4 is evaluated by accumulating the layerwise runtime. The reported power is the total board power that measured by actual testing on the FPGA board. Comparing the RetinaNet at the input scale of 600 to LW-RetinaNet-v3, LW-RetinaNet-v3 achieves an 0.3% mAP improvement over the original RetinaNet for the same runtime, and also is 1.8x more energy-efficient. One can observe that the actual runtime is approximately proportional to the FLOP count in Table 4, which also validates the feasibility of choosing FLOP count as the indicator to optimize the heavy FLOP layers for speedup.

As any detection methods with FPN structure can result in an imbalanced FLOP distribution, the proposed method can be potentially applied to any such kind of detection network for a better FLOP-mAP trade-off with more energy-efficient edge inference.

V. CONCLUSION

In this paper, we present a light-weight RetinaNet model that has a constantly better FLOP-mAP trade-off curve (linear degradation) than a naive input image scaling approach (polynomial degradation). The key is to substitute the heaviest bottleneck layer of blockwise-FLOP-imbalance RetinaNet

TABLE IV
COMPARISON OF ORIGINAL RETINANET AND PROPOSED LIGHT-WEIGHT RETINANET.

	scale	mAP	AP50	AP75	APS	APM	APL	GFLOPs	ratio	runtime(s)	power efficiency (μJ/pixel)
RetinaNet	800	35.7	55	38.5	18.9	38.9	46.3	156	0	1.7	74
RetinaNet	700	35.1	54.2	37.7	18	39.3	46.4	119	1.3x	1.3	74
RetinaNet	600	34.3	53.2	36.9	16.2	37.4	47.4	88	1.8x	0.9	70
LW-RetinaNet-v1	800	35.4	54.4	38.2	18.3	38.7	46	135	1.1x	1.5	66
LW-RetinaNet-v2	800	35.1	54.3	37.7	17.9	38.4	45.7	114	1.4x	1.2	53
LW-RetinaNet-v3	800	34.6	53.1	37.3	15.7	38.7	44.6	89	1.8x	0.9	39

with simplified building blocks, while keeping the rest of the network untouched. Experiment results show that, at a 1.8x FLOP reduction point, the light-weight RetinaNet achieves 0.3% mAP improvement and 1.8x more energy-efficiency on an FPGA-based edge node. The proposed method can be potentially applied to any FPN-based detection network that has imbalanced blockwise FLOP distribution for an improved FLOP-mAP trade-off, with more energy-efficient inference at the edge.

ACKNOWLEDGMENT

This work is supported by an NSF grant (IIS/CPS-1652038) and an unrestricted gift from Radius AI, Inc. The computing infrastructure used in this work is supported by an NFS grant (CNS-1629888). The Arria 10 GX FPGA Development Kits used for this research was donated by Intel FPGA University Program. We thank Dr. Aykut Dengi and Bobby Chowdary from Radius AI, Inc. for fruitful research discussions.

REFERENCES

[1] S. Biookaghazadeh, M. Zhao, and F. Ren. Are fpga suitable for edge computing? In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[2] A. Dua. Hardware acceleration of video analytics using opencl.

[3] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[4] R. Girshick, J. Donahue, T. Darrell, et al. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[5] R. Girshick, I. Radosavovic, G. Gkioxari, et al. Detectron. https://github.com/facebookresearch/detectron, 2018.

[6] P. Goyal, P. Dollár, R. Girshick, et al. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[7] K. He, G. Gkioxari, P. Dollár, et al. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] A. G Howard, M. Zhu, B. Chen, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[10] J. Huang, V. Rathod, C. Sun, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[11] T. Lin, P. Dollár, R. Girshick, et al. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[12] T. Lin, P. Goyal, R. Girshick, et al. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[13] T.-Y. Lin, M. Maire, S. Belongie, et al. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[14] W. Liu, D. Anguelov, D. Erhan, et al. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[15] Z. Liu, M. Sun, Y. Zhou, et al. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[16] Radius AI. Changing the landscape of the c-store market, 2019.

[17] J. Redmon, S. Divvala, R. Girshick, et al. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[18] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[19] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[20] S. Ren, K. He, R. Girshick, et al. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, et al. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.