A Compression-Compilation Co-Design Framework Towards Real-Time Object Detection on Mobile Devices

Yuxuan Cai[1]*, Geng Yuan[1]*, Hongjia Li[1]*, Wei Niu[2], Yanyu Li[1], Xulong Tang[3], Bin Ren[2], Yanzhi Wang[1]

[1]Northeastern University
[2]College of William and Mary
[3]University of Pittsburgh
[1]{cai.yuxu, yuan.geng, li.hongjia, li.yanyu, yanz.wang}@northeastern.edu,
[2]wniu@email.wm.edu, [2]bren@cs.wm.edu, [3]tax6@pitt.edu

Abstract

The rapid development and wide utilization of object detection techniques have aroused requirements for both accuracy and speed of object detectors. In this work, we propose a compression-compilation co-design framework to achieve real-time YOLOv4 inference on mobile devices. We propose a novel fine-grained structured pruning, which maintain high accuracy while achieving high hardware parallelism. Our pruned YOLOv4 achieves 48.9 mAP and 17 FPS inference speed on an off-the-shelf Samsung Galaxy S20 smartphone, which is 5.5× faster than the original state-of-the-art detector YOLOv4.

Introduction

Object detection is widely adopted in numerous computer vision tasks with a wide range of applications, such as autonomous driving, robot vision and human-computer interaction. However, the state-of-the-art object detection works are either accuracy-oriented using a large model size (Liu et al. 2016; Bochkovskiy, Wang, and Liao 2020) with high inference latency or speed-oriented using a lightweight model but sacrificing accuracy (Sandler et al. 2018; Huang, Pedoeem, and Chen 2018). None of them can satisfy the the accuracy and latency demands of practical applications on mobile devices simultaneously.

Weight pruning is a promising candidate solution for this issue, which has been demonstrated as an effectively approach to reduce extensive computation and memory intensity of DNN models (He et al. 2019). Generally, unstructured pruning (Han et al. 2015) and structured pruning (Wen et al. 2016) are the two main trendy schemes of weight pruning. However, they are either hard to achieve hardware acceleration or suffering from notable accuracy degradation (Ma et al. 2019). The pattern-based pruning (Ma et al. 2020) is proposed to overcome these issues by incorporating fine-grained unstructured pruning in a hardware-aware fashion. But it is only applicable to convolutional (CONV) layers with 3×3 kernels, significantly limiting its deployment in object detection tasks.

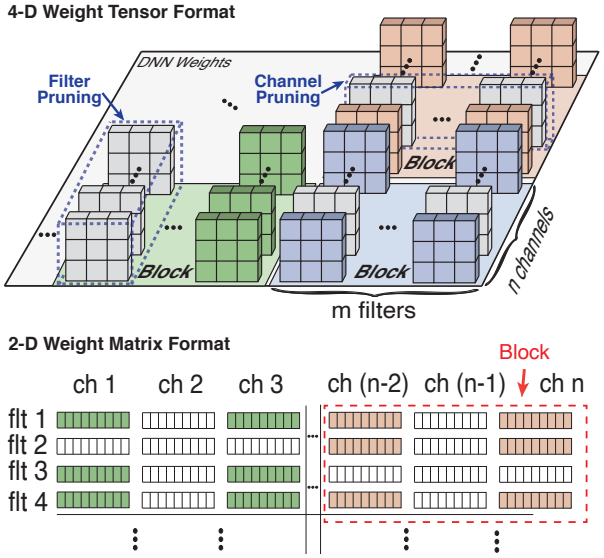

Figure 1: Fine-grained structured pruning for CONV and FC layers using 4D tensor and 2D matrix representation.

In this paper, we propose compression-compilation co-design framework, which can achieve real-time and high accuracy object YOLOv4 detection inference on mobile platforms.

Here is an overview of our framework:

- We use a fine-grained structured pruning scheme as the model compression technique, which combines the advantage of high accuracy and the capability of achieving high hardware-parallelism.
- Under our compiler-assisted optimization, with comparable accuracy to state-of-the-art object detectors, our proposed framework achieves real-time object detection on mobile devices.

*These Authors contributed equally.

Framework Design

Fine-grained Structured Pruning

Figure 1 shows the fine-grained structured pruning scheme in 4D weight tensor format and 2D weight matrix format. Fine-grained Structured Pruning is an extension of the coarse-grained structured pruning that prunes whole filters/channels, or rows/columns in matrix-based computation. The entire weight matrix is divided into a number of equal-sized blocks, then the entire column(s) and/or row(s) of weights are pruned within each block. Compared to the coarse-grained structured pruning, block-based pruning provides a finer pruning granularity to better preserve the DNN model accuracy. And a high hardware parallelism is also achieved by maintaining an appropriate structural regularity according to the block-punched pruning constraints. It is particularly suitable for high-efficient DNN inference on resource-limited mobile devices. In addition to the 3×3 CONV layer, it can also be mapped to other types of DNN layers, such as 1×1 CONV layer and FC layer.

As our block-based pruning divides the weight matrix into blocks, the block size affects both the accuracy and the hardware acceleration. A smaller block size can provide higher structural flexibility but at the cost of reduced speed. While larger block size can better leverage the hardware parallelism, but it may cause more severe accuracy loss. We show the results of accuracy (mAP) vs. speed (FPS) using different block size in Figure 2.

Compiler Optimization

Our fine-grained structured pruning relies on the compiler optimizations to extract the fine-grained structure information in pruned models and generate optimized code that can run on mobile CPU and GPU efficiently. It first compacts the model storage with a novel compression format called BCS (Blocked Compressed Storage format), and then performs computation reorder to reduce the branches within each thread and eliminate the load imbalance among threads.

BCS stores non-zero weights only as Compressed Sparse Row format (CSR) with an even better compression ratio by further compressing the index with a hierarchical structure.

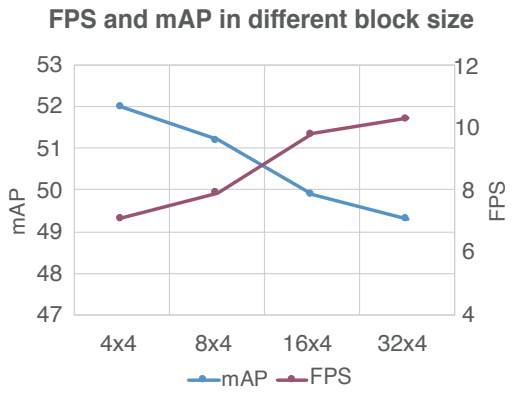

Figure 2: Accuracy (mAP) vs. speed (FPS) of different block size pruning results.

Figure 3: Real-time detection on Samsung S20 smartphone.

Traditional CSR needs to store each non-zero weight with an explicit column index. Our proposed block-based pruning preserves non-zero weights in identical columns in each block, thus leading to many repeated column indices if we use CSR. BCS eliminates this redundancy with a hierarchical compression on the column index only.

Moreover, our compiler employs a similar optimization flow (i.e., model compaction and computation reorder and other optimizations) to support all compiler optimizations for pattern-based pruning as PatDNN (Niu et al. 2020).

Experiments and Demonstrations

Experimental Setup

We evaluate our framework on an off-the-shelf Samsung Galaxy S20 smartphone, which has a Qualcomm Snapdragon 865 Octa-core CPU and a Qualcomm Adreno 650 GPU. Each test runs on 50 different input frames (images), with the average speed results reported. Our work is derived based on YOLOv4, with 320×320 input size. The MS COCO dataset is used for training and evaluation. We denote mAP as the Average Precision under IoU 0.5 threshold and AP@[.5:.95] as the Average Precision under IoU from 0.5 to 0.95.

Evaluation of Fine-grained Structured Pruning

We first evaluate the accuracy and compression rate of our proposed fine-grained structured pruning. As mentioned above, block size affects both accuracy and hardware acceleration performance. We adopt 8×4 as our block size, i.e. 4 consecutive channels of 8 consecutive filters. The details of the impact of different block sizes are discussed in ablation study . The original YOLOv4 model contains

#Weights	#Weights Comp. Rate	#FLOPs	mAP	AP@[.5:.95]	FPS
64.36M	1×	35.8G	57.3	38.2	3.5
16.11M	3.99×	10.48G	55.1	36.5	7.3
8.04M	8.09×	6.33G	51.4	33.3	11.5
6.37M	10.1×	5.48G	50.9	32.8	13
4.59M	14.02×	3.95G	48.9	31.6	17

Table 1: Accuracy and speed under different compression rates.

Approach	Input Size	backbone	#Weights	#FLOPs	mAP	AP@[.5:.95]	FPS
CenterNet-DLA (Duan et al.)	512	DLA34	16.9M	52.58G	57.1	39.2	1.9
CornerNet-Squeeze (Law et al.)	511	-	31.77M	150.15G	-	34.4	0.3
SSD (Liu et al.)	300	VGG16	26.29M	62.8G	43.1	25.1	4.2
MobileNetv2-SSDLite (Sandler et al.)	300	MobileNetv2	3.38M	1.36G	-	22.1	41
YOLOv4 (Bochkovskiy, Wang, and Liao)	320	CSPDarknet53	64.36M	35.5G	57.3	38.2	3.5
YOLO-Lite (Huang, Pedoeem, and Chen)	224	-	0.6M	1.0G	-	12.26	36
YOLOv3-tiny (Redmon and Farhadi)	320	Tiny Darknet	8.85M	3.3G	29	14	14
YOLOv4-tiny (Bochkovskiy, Wang, and Liao)	320	Tiny Darknet	6.06M	4.11G	40.2	-	11
Ours	320	CSPDarknet53	4.59M	3.95G	**48.9**	**31.6**	**17**

Table 2: Accuracy (mAP) and speed (FPS) comparison with other object detection approaches.

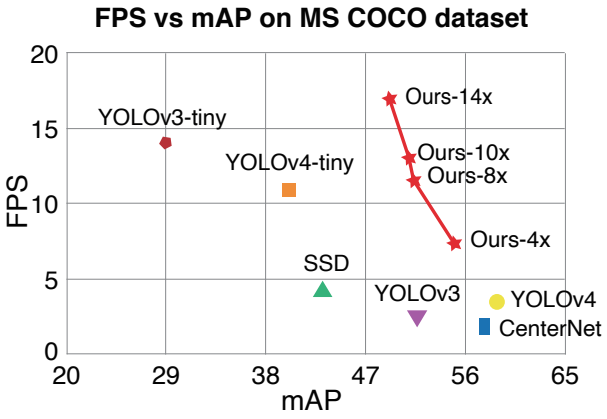

Figure 4: The accuracy (mAP) and speed (FPS) comparison of our framework under different compression rate and different approaches.

64.36M weights and requires 35.8G floating-point operations (FLOPs). As shown in Table 1, by applying our fine-grained structured pruning, we achieve the compression rate up to 14× (in weights) with 48.9 mAP. The weight number decreases to 4.59M and FLOPs is reduced to 3.59G. With 92.87% weights and 88.97% FLOPs reduced, our model still maintains a decent accuracy, with only 8.3 mAP loss.

As shown in Table 1, we reduce 92.87% of weights and 88.97% of computations with only 8.3% mAP loss on COCO 2014 dataset. We also demonstrate when we reduce the compression rate, we can reach the mAP of 0.509 and 0.514 under 8× and 10× compression rate.

Evaluation of Our Framework

Field tests are made on a Samsung S20 smartphone under different scenarios, as shown in Figure 3. Demo videos can be found on the website[1][2].

And to further validate the effectiveness of our framework, we compare our pruned YOLOv4 with several representative works. To make fair comparisons, all the results (including the reference works) are evaluated under our compiler optimizations. As shown in Table 2, by applying our block-punched pruning, compared to original

[1]https://youtu.be/fDOdMReskjQ

[2]https://www.bilibili.com/video/BV1bz4y1y7CR/

Pruning method	#Weights	#FLOPs	mAP	FPS
Evenly Prune	10.38M	6.2G	50.5	8
Unevenly Prune	8.04M	6.2G	51.4	11.5

Table 3: Evenly Prune vs. Unevenly Prune.

YOLOv4 model, we achieve the compression rate up to 14× (in weights) with 48.9 mAP. The number of weights decreases to 4.59M and FLOPs is reduced to 3.59G. Under a similar number of computations and even having a smaller model size, our pruned YOLOv4 consistently outperforms YOLOv3-tiny and YOLOv4-tiny in terms of mAP and FPS. This indicates our proposed block-punched pruning is a more desired method to achieve a smaller model size while maintaining the mAP compared to training a small model from scratch. Compared with the full-size object detectors, our work has fairly accuracy but much faster inference speed (5.5× faster than state-of-the-art detector YOLOv4). The results also indicate our GPU-CPU collaborative computation scheme effectively accelerates the inference speed and improves FPS.

Figure 4 demonstrates the mAP and FPS of our pruned YOLOv4 under different compression rates and the results are compared with representative reference works. Our optimized model lies in top right of the figure, and outperforms YOLOv3, SSD, YOLOv3-tiny and YOLOv4-tiny in both accuracy and speed. Unlike the lightweight approaches, which simply trade the mAP for FPS, our approach provides a Pareto Optimality that maintains both the mAP and FPS.

Ablation Study

Ablation Study on pattern-based pruning. Despite the promising performance that pattern-based pruning can achieve, it has the restriction of kernel size to be 3, while in non-3×3 layers it cannot be applied. Unfortunately, in object detection approaches such as YOLOv4, the ratio of 3×3 CONV layers is 83.31% in total weights, which limits the highest compression rate of pattern-based pruning to 5.99×. Therefore, we compare pattern-based pruning and our block-punched pruning scheme under compression rate of 2×, 3×, 4×, 5×, respectively. Additionally, we demonstrate the result under our block-punched pruning scheme with 8×, 10× and 14 × compression rate. In Figure 5, we plot the mAP curve and FPS bar under different compres-

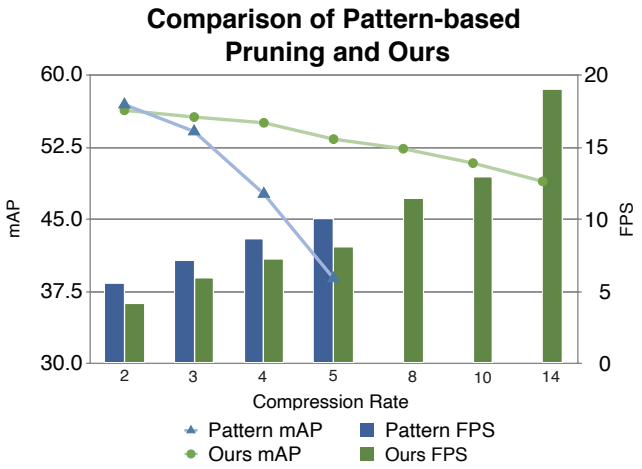

Figure 5: mAP and FPS comparison of pattern-based pruning and ours.

sion rate. We can see when the compression rate is below 3× pattern-based pruning has higher accuracy as it is more flexible. When the compression rate increases, for pattern-based pruning, we have to prune more weights in each 3×3 CONV layer, because non-3×3 layers can not be pruned. The extremely high layer-wise prune ratio results in a sharp drop down of the curve. Pattern-based pruning has higher inference speed compared to our block-punched pruning. However, the compression rate ceiling limits the inference speed. Ours scheme can reach higher speed when compression rate increases, and ours do not suffer from sharply accuracy drop down.

Ablation study on layer-wise compression rate between different kernel size. YOLOv4 contains only 3×3 kernel size and 1×1 kernel size in CONV layers. We believe these 2 types of CONV layers have different levels of sensitivity in pruning process, therefore we conduct 2 groups of experiments. Under the same number of FLOPs, we evenly prune all the layers in one group. And in another group, the compression rate of 3×3 CONV layers is 1.15× higher than in 1×1 CONV layers. As shown in Table 3, the evenly pruned model exhibits lower accuracy and lower inference speed than the unevenly pruned model. Since 3×3 CONV layers contributes 81.4% of the total FLOPs, it can be concluded that compression rates in these layers illustrate a higher impact on the overall performance.

References

Bochkovskiy, A.; Wang, C.-Y.; and Liao, H.-Y. M. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934* .

Duan, K.; Bai, S.; Xie, L.; Qi, H.; Huang, Q.; and Tian, Q. 2019. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 6569–6578.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems (NeurIPS)*.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 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)*.

Huang, R.; Pedoeem, J.; and Chen, C. 2018. YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In *2018 IEEE International Conference on Big Data (Big Data)*, 2503–2510. IEEE.

Law, H.; Teng, Y.; Russakovsky, O.; and Deng, J. 2019. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900* .

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*.

Ma, X.; Guo, F.-M.; Niu, W.; Lin, X.; Tang, J.; Ma, K.; Ren, B.; and Wang, Y. 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)*.

Ma, X.; Lin, S.; Ye, S.; He, Z.; Zhang, L.; Yuan, G.; Tan, S. H.; Li, Z.; Fan, D.; Qian, X.; Lin, X.; Ma, K.; and Wang, Y. 2019. Non-Structured DNN Weight Pruning – Is It Beneficial in Any Platform?

Niu, W.; Ma, X.; Lin, S.; Wang, S.; Qian, X.; Lin, X.; Wang, Y.; and Ren, B. 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)*.

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

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.