Abstract—Object detection plays an important role in self-driving cars for security development. However, mobile systems on self-driving cars with limited computation resources lead to difficulties for object detection. To facilitate this, we propose a compiler-aware neural pruning search framework to achieve high-speed inference on autonomous vehicles for 2D and 3D object detection. The framework automatically searches the pruning scheme and rate for each layer to find a best-suited pruning for optimizing detection accuracy and speed performance under compiler optimization. Our experiments demonstrate that for the first time, the proposed method achieves (close-to) real-time, 55ms and 99ms inference times for YOLOv4 based 2D object detection and PointPillars based 3D detection, respectively, on an off-the-shelf mobile phone with minor (or no) accuracy loss.

I. INTRODUCTION

As the rapid development of the autonomous vehicles, object detection including 2D and 3D detection is one of the most important prerequisites to autonomous navigation. It is essential to implement real-time object detection on autonomous vehicles due to security considerations. However, as 2D and 3D detection are implemented with deep neural networks (DNNs) such as YOLO [2] and PointPillars [9], respectively, with tremendous memory and computation requirements, it is challenging to achieve real-time on autonomous vehicles with limited memory and computation resources.

To achieve real-time object detection on edge devices with limited resources, we propose neural pruning search with compiler optimization to implement real-time 2D detection with YOLO [2] and 3D object detection with PointPillars [9] on mobile devices. We summarize our contribution as follows,

- We propose to perform a novel compiler-aware neural pruning search with Bayesian optimization (BO), automatically determining the pruning scheme and rate (including bypass) for each individual layer. The objective is to maximize accuracy satisfying an inference latency constraint on the target mobile device.

- We can achieve (close-to) real-time, 55ms and 99ms inference times for YOLOv4 based 2D detection and PointPillars based 3D detection, respectively, on an off-the-shelf mobile phone with minor (or no) accuracy loss. Our method on 2D detection notably outperforms other acceleration frameworks such as TVM [3] and MNN [1], while we are the first to support 3D detection on mobile.

II. AUTOMATIC NEURAL PRUNING SEARCH

The framework consists of two basic components: a controller and an evaluator. The controller first generates various pruning proposals from the search space. Then the evaluator evaluates their detection accuracy and speed performance. Based on the performance, the evaluator provides guidance for controller about what a satisfying pruning proposal looks like. Next the controller generates new pruning proposals with the guidance. After iterations, the controller outputs the best pruning proposal with desirable detection performance while satisfying the real-time requirement.

A. Controller

The controller generates pruning proposals from the search space. Each pruning proposal consists of the pruning scheme and rate for each layer of the model, as shown in Tab. I.

Per-layer pruning schemes: The controller can choose from filter (channel) pruning [14], pattern-based pruning [12] and block-based pruning [5] for each layer.

Per-layer pruning rate: We can choose from the list \{1×, 2×, 2.5×, 3×, 5×, 7×, 10×, skip\}, where 1× means the layer is not pruned, and “skip” means bypassing this layer.

1) Evaluation with BO: To deal with the non-continuous and graph-like pruning constraints, we use BO to accelerate evaluation. As shown in Algorithm I, given a proposed pool from the controller, we first adopt BO to select a part of proposals with potentially better performance and evaluate their accurate detection and speed performance, while the rest potentially weak proposals are not evaluated. Thus, we reduce the number of actual evaluated proposals.

| TABLE I | SEARCH SPACE FOR EACH DNN LAYER |
|---------|---------------------------------|
| Pruning scheme | \{Filter [14], Pattern-based [12], Block-based [5]\} |
| Pruning rate | \{ 1×, 2×, 2.5×, 3×, 5×, 7×, 10×, skip \} |
After selecting $B$ pruning proposals from the pool, we evaluate their performance using magnitude based framework \cite{1} following their pruning proposals for each layer.

2) **Gradients Guidance:** To guide the proposal updating, we employ the derivatives of the GP predictive mean with reference to the number of nodes in the graph. Basically, positive gradients show that the node is beneficial to improve the reward, while negative gradients mean that the node decreases the performance and it should be replaced. To make the gradients more illustrative, we transform the gradients into a probability distribution (replacement probability) using a sigmoid transformation on the gradient of the positives and then normalize them. Thus, negative gradients lead to high replacement probabilities. To summarize, the evaluator provides the gradient guidance including the best evaluated pruning proposal and its corresponding replacement probability obtained from its gradients.

### III. Experimental Results

For 2D object detection, we use a YOLOv4 \cite{2} model as starting point and test on COCO dataset \cite{10}. For 3D detection, we employ the PointPillars as starting point \cite{9} and test on KITTI dataset \cite{6}. All the acceleration results are tested on the mobile GPU of a Samsung Galaxy S20 smartphone.

For 2D detection, as shown in Fig. 1 on mobile GPU, our method achieves 5.18× inference acceleration (285.7ms vs. 55.2ms) compared with the original model. Compared with other pruning schemes, under the same pruning rate, our method is a bit slower than filter pruning on mobile GPU but achieves much higher accuracy (49.3 vs. 25.2 in mAP). With slightly lower accuracy, our method is 1.79× faster than unstructured pruning. We achieve faster speed compared with other acceleration frameworks such as MNN \cite{1}. For 3D detection with point clouds, we start from PointPillars and test with different different grid sizes (0.16m and 0.24m). The real-time requirements are set to 200ms for 0.16m grid size and 100ms for 0.24m. As shown in Tab. 1 and Fig. 2, we can observe that, for the same grid size, our method can significantly reduce the parameter count and computation, thus satisfying the real-time requirement, while achieving state-of-the-art detection performance. For a grid size of 0.24m, under the same overall pring ratio (86%), the proposed method can achieve the best detection performance compared with other methods with the same pruning scheme for each layer, demonstrating the advantages of using flexible pruning scheme for each layer. Besides, with compiler optimization, filter pruning is the fastest but suffers from obvious detection performance degradation. The proposed method can process one LiDAR image within 99ms with the highest precision, achieving (close-to) real-time inference on mobile.

### IV. Acknowledgements

This project is partly supported by National Science Foundation (NSF) under grants CNS-1932351, CNS-1909172, and CMMI-2013067, Army Research Office (ARO) Young Investigator Program 76598CSYP, a grant from Semiconductor Research Corporation (SRC), and Jeffress Trust Awards in Interdisciplinary Research. Any opinions, findings, and conclusions or recommendations in this material are those of the authors and do not necessarily reflect the views of NSF, ARO, SRC, or Thomas F. and Kate Miller Jeffress Memorial Trust.

### REFERENCES

[1] https://github.com/alibaba/MNN.

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” arXiv:2004.10934, 2020.

[3] T. Chen, T. Moreau et al., “Tvm: An automated end-to-end optimizing compiler for deep learning,” in USENIX, 2018, pp. 578–594.

[4] Y. Chen, A. Huang et al., “Bayesian optimization in alphago,” arXiv:1812.06855, 2018.

[5] P. Dong, S. Wang et al., “Rtmobile: Beyond real-time mobile acceleration of rns for speech recognition,” arXiv:2002.11474, 2020.

[6] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in CVPR, 2012.

[7] S. Han, J. Pool et al., “Learning both weights and connections for efficient neural network,” in NeurIPS, 2015, pp. 1135–1143.

[8] Y. He, P. Liu et al., “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in CVPR, 2019.

[9] A. H. Lang, S. Vora et al., “Pointpillars: Fast encoders for object detection from point clouds,” in CVPR, 2019, pp. 12697–12705.

[10] T.-Y. Lin et al., “Microsoft coco: Common objects in context,” in ECCV.

[11] X. Mu et al., “Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices,” in AAAI, 2020.

[12] W. Niu et al., “Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning,” in arXiv:2001.00135, 2020.

[13] N. Shervashidze, P. Schweitzer et al., “Weisfeiler-levens graph kernels,” Journal of Machine Learning Research, vol. 12, no. 77, 2011.

[14] Z. Zhuang, M. Tan et al., “Discrimination-aware channel pruning for deep neural networks,” in NeurIPS, 2018, pp. 875–886.