Brief Industry Paper: Towards Real-Time 3D Object Detection for Autonomous Vehicles with Pruning Search

^{1}Pu Zhao, ^{2}Wei Niu, ^{1}Geng Yuan, ^{1}Yuxuan Cai, ^{3}Hsin-Hsuan Sung, ^{4}Shaoshan Liu,
^{5}Sijia Liu, ^{3}Xipeng Shen, ^{2}Bin Ren, ^{1}Yanzhi Wang, ^{1}Xue Lin
^{1}*Northeastern University, Boston, MA*
^{2}*William & Mary, Williamsburg, VA*
^{3}*North Carolina State University, NC*
^{4}*PerceptIn, CA*
^{5}*Michigan State University, MI*

***Abstract*—In autonomous driving, 3D object detection is essential as it provides basic knowledge about the environment. However, as deep learning based 3D detection methods are usually computation intensive, it is challenging to support real-time 3D object detection on edge-computing devices in self-driving cars with limited computation and memory resources. To facilitate this, we propose a compiler-aware pruning search framework, to achieve real-time inference of 3D object detection on the resource-limited mobile devices. Specifically, a generator is applied to sample better pruning proposals in the search space based on current proposals with their performance, and an evaluator is adopted to evaluate the sampled pruning proposal performance. To accelerate the search, the evaluator employs Bayesian optimization with an ensemble of neural predictors. We demonstrate in experiments that the pruning search framework can achieve real-time 3D object detection on mobile (Samsung Galaxy S20 phone) with state-of-the-art detection performance.**

***Index Terms*—3D object detection, real-time, point cloud**

I. Introduction

As the rapid development of autonomous vehicles to self-drive without human intervention, object detection (especially 3D detection to deal with LiDAR data) serves as a fundamental prerequisite for autonomous navigation. 3D detection can extract the desirable knowledge about its environment from 3D point clouds of LiDAR sensors, thus enabling high-level computations and optimizations for auto-driving.

Due to the instantaneously interaction requirement with the environment in auto-driving, it is essential to implement real-time 3D object detection on autonomous vehicles. However, the current deep neural networks (DNNs) based 3D object detectors usually cost tremendous memory and computation resources, leading to difficulties for real-time implementation, especially on autonomous vehicles with limited hardware resource. Though more powerful high-end GPUs can be adopted for this task, they usually result in significant increasing price and power consumption. Thus it is desirable to facilitate the real-time 3D detection deployment on autonomous cars.

To reduce the DNN model size and computations, DNN weight pruning [1], [2] has shown great advantages to remove redundancy in the model, therefore reducing storage/computation cost and accelerating inference. There are *unstructured pruning* scheme [2]–[4] to remove arbitrary weight, *coarse-grained structured pruning* scheme [1], [4]–[7] to eliminate whole filters/channels, and *fine-grained structured pruning* [8]–[10] to assign different pruning patterns to convolutional (CONV) kernels. Though unstructured pruning can achieve high accuracy, the arbitrary pruned irregular weights limited hardware parallelism, leading to difficulties for inference acceleration. Compared with unstructured pruning, structured pruning can achieve higher hardware parallelism and mobile inference acceleration, assisted by the compiler-level code generation and optimization techniques [9], with competitive classification/detection performance.

Though compiler optimization can support various structured pruning (sparsity) schemes with notable mobile acceleration performance, we found that different sparsity schemes lead to different accuracy and acceleration performance with compiler optimization. For the specific 3D detection problem, it is still questionable to adopt which sparsity scheme with which pruning rate to satisfy the accuracy and real-time requirements. To find the pruning solution, motivated by the idea of Neural Architecture Search (NAS) [11], [12], we propose a compiler-aware pruning search framework to automatically determine the pruning scheme and pruning rate for each individual layer. The *objective* is to maximize accuracy with an inference speed/latency constraint on the target mobile device. Different from previous work with fixed pruning scheme for all layers, our work can have different pruning schemes and rates for different layers in the model. We summarize our contribution as follows,

- We incorporate the overall DNN latency constraint into automatic pruning search process to satisfy a predefined real-time requirement.
- Our framework configures different pruning schemes and pruning rates for different layers which is different from previous works with fixed pruning scheme for all layers.

- We adopt an ensemble of neural predictors and Bayesian optimization (BO) to reduce the number of evaluated pruning proposals, leading to less searching efforts.
- We can achieve (close-to) real-time (98ms) 3D detection with PointPillars, on an off-the-shelf mobile phone with minor (or no) accuracy loss.

II. Background and Related Work

A. 3D Object Detection

3D object detection detects objects with point clouds from LiDAR sensors. PointPillars [13] is a popular 3D detection method with three main stages: (1) A feature encoder network to convert a point cloud to a sparse pseudo-image; (2) a 2D CONV backbone to transform the pseudo-image into high-level representation; and (3) a detection head to regress 3D boxes. Besides PointPillars, there are various 3D detection methods such as SECOND [14] and Point-GNN [15]. We mainly focus on PointPillars as we found that PointPillars is the only one whihc can run on mobile while others are not available on mobile since their special structures to deal with sparse data are not supported by mobile compiler. Besides, PointPillars costs less computations than others with faster inference speed on server GPUs (e.g., 25ms for PointPillars v.s. 600ms for Point-GNN).

B. Weight Pruning Schemes

Previous weight pruning work can be categorized according to pruning scheme: unstructured pruning [2], [3], [16], coarse-grained structured pruning [1], [4]–[6], and fine-grained structured pruning including pattern [8] and block [10] pruning.

Unstructured pruning [3], [16] removes weights at arbitrary positions, leading to irregular sparse weight matrix with indices, incurring damages to the parallel implementations and acceleration performance on hardware. Different from unstructured pruning, coarse-grained structured pruning [4], [5] removes the whole filters/channels to maintain model structure with high regularity for efficient hardware parallel implementation, at the cost of certain obvious accuracy degradation. To overcome the disadvantages, fine-grained structured pruning [8], [9] follows a pruning pattern (chosen from a predefined library) to prune each CONV kernel, where the predefined patterns have been optimized with compiler optimizations for mobile acceleration. Fine-grained structured pruning can achieve high accuracy due to the flexibility with different patterns, and high hardware parallelism (and mobile acceleration) with compiler-based code generation and optimization.

III. Automatic Neural Pruning Search

We show the framework in Fig. 1, consisting of two basic components: a *generator* and an *evaluator*. Given the search space, the generator first generates or samples various *pruning proposals*. Then the evaluator evaluates their detection accuracy and speed performance, and feeds them back to the generator. Next the generator samples new pruning proposals based on existing proposals' performance. After iterations, the framework can obtain the final pruning proposal with satisfying detection accuracy and speed performance.

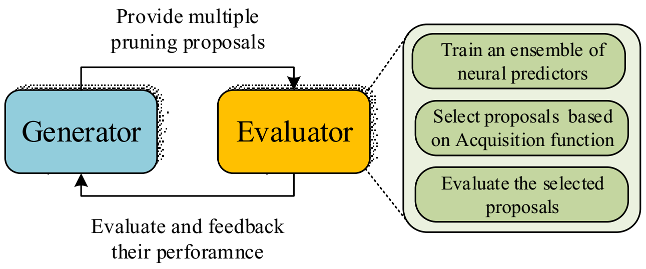
Provide multiple pruning proposals
Generator
Evaluator
Evaluate and feedback their perforamnce
Train an ensemble of neural predictors
Select proposals based on Acquisition function
Evaluate the selected proposals

Fig. 1. Automatic network pruning search framework

TABLE I
SEARCH SPACE FOR EACH DNN LAYER

Pruning scheme	{Filter [18], Pattern-based [9], Block-based [10]}
Pruning rate	{ 1×, 2×, 3×, 5×, 7×, 10×, 15× }

In each iteration, the evaluator first trains an ensemble of neural predictors and then selects proposals based on their acquisition function values enabled by the predictor ensemble. Next the selected proposals are evaluated to obtain their performance while the rest unselected proposals are not evaluated, thus reducing evaluation time and efforts.

After the framework finishes and outputs a final pruning proposal, we further apply ADMM pruning [17] to perform an enhanced pruning following the best proposal. Compared with the simple magnitude pruning [3] method applied during evaluation for time-saving, ADMM usually outperforms magnitude pruning in terms of accuracy with an increased complexity, that is why we only adopt it for the final proposal.

A. Generator

The generator samples *pruning proposals* from the search space. Each pruning proposal is a directed graph consisting of the layer-wise pruning scheme and layer-wise pruning rate. For example, it has 20 nodes for a 10-layer DNN model.

1) Proposal Formulation (Search Space): Each pruning proposal contains the pruning scheme and pruning rate for each layer of the model, as shown in Tab. I.

Per-layer pruning schemes: The generator can choose from filter (channel) pruning [18], pattern-based pruning [8] and block-based pruning [10] for each layer. As different layers may have different best-suited pruning schemes, the generator can choose different pruning schemes for different layers, also supported by our compiler code generation.

Per-layer pruning rate: The pruning rate is the rate between the number of original parameters and that of left parameters after pruning. We can choose from the list $\{1\times, 2\times, 3\times, 5\times, 7\times, 10\times, 15\times\}$, where $1\times$ means the layer is not pruned (i.e., bypassing this layer).

2) Proposal Updating: The generator keeps a record of all evaluated pruning proposals with their evaluation performance. To generate new pruning proposals, it mutate the proposals with the best evaluation performance in the records by randomly changing one pruning scheme or one pruning rate of one layer. More specifically, it first selects K proposals with

Algorithm 1 Evaluation with predictor ensemble & BO

```
Input: Observation data $\mathcal{D}$, BO batch size $B$, BO acquisition function $\phi(\cdot)$
Output: The best pruning proposal $g$
for steps do
    Generate a pool of candidate pruning proposals $\mathcal{G}_c$;
    Train an ensemble of neural predictors with $\mathcal{D}$;
    Select $\{\hat{g}^i\}_{i=1}^B = \arg\max_{g \in \mathcal{G}_c} \phi(g)$;
    Evaluate the proposal and obtain reward $\{r^i\}_{i=1}^B$ of $\{\hat{g}^i\}_{i=1}^B$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\{\hat{g}^i\}_{i=1}^B, \{r^i\}_{i=1}^B)$;
end for
```

the highest evaluation performance, and mutates each of them iteratively until it gets C new proposals.

3) Proposal encoding: As pruning proposals are basically graphs, special attention is required for the proposal representation. Different from traditional representations with an adjacency matrix for graphs, we adopt the pruning encoding to encode each proposal with a vector of binary values. There is a binary feature for each possible node in each layer, denoting whether the node (pruning scheme or pruning rate of certain layer) is adopted or not. To encode a proposal, we simply check which pruning scheme or rate for each layer is applied, and set the corresponding features to 1s. This simple proposal encoding can help with proposal evaluation.

B. Evaluator

The evaluator needs to evaluate pruning proposal performance. We define the performance measurement (reward) as:

$$m = V - \alpha \cdot \max(0, r - R), \tag{1}$$

where V is the validation mean average precision (mAP) of the model, r is the model inference latency, which is actually measured on a mobile device with compiler code optimization and generation for inference acceleration. R is the real-time requirement threshold. Generally, satisfying real-time requirement ($r < R$) with high mAP leads to high m. Otherwise if the real-time requirement is violated, m is small.

1) Fast Evaluation with BO: As it incurs large time cost to evaluate the performance of each pruning proposal (including pruning and retraining the model with multiple epochs), we adopt Bayesian optimization (BO) [19] to accelerate evaluation. The generator provides C pruning proposals, and the evaluator first use BO to select B proposal with potentially better performance. Next the evaluator measure the accurate accuracy and speed performance of the selected proposals while the rest unselected proposals are not evaluated. Thus, the number of actual evaluated proposals is reduced.

In general, there are two main components in BO including training an ensemble of neural predictors and selecting proposal based on acquisition function values enabled by the predictor ensemble. To make use of BO, the ensemble of neural predictors provides an accuracy prediction with its corresponding uncertainty estimate for an unseen pruning proposal. Then BO is able to choose the proposal which maximizes the acquisition function. We show the full algorithm in Algorithm 1 and specify the two components in the following.

2) Ensemble of Neural Predictors: We use a neural network repeatedly trained on the current set of evaluated pruning proposals with their evaluation performance as a neural predictor to predict the reward (incorporating the accuracy and speed performance) of unseen pruning proposals. The neural network is a sequential fully-connected network with 8 layers of width 30 trained by the Adam optimizer with a learning rate of 0.01. Note that it does not cost much predictor training efforts due to their simple architectures and parallel training.

For the loss function in neural predictors, mean absolute percentage error (MAPE) is adopted as it can give a higher weight to pruning proposals with higher evaluation performance:

$$\mathcal{L}(m_{\text{pred}}, m_{\text{true}}) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{m_{\text{pred}}^{(i)} - m_{\text{UB}}}{m_{\text{true}}^{(i)} - m_{\text{UB}}} - 1 \right|, \tag{2}$$

where $m_{\text{pred}}^{(i)}$ and $m_{\text{true}}^{(i)}$ are the predicted and true values of the reward for the i-th proposal in a batch, and m_{UB} is a global upper bound on the maximum true reward.

To incorporate BO, it also needs an uncertainty estimate for the prediction. So we adopt an ensemble of neural predictors to provide the uncertainty estimate. More specifically, we train P neural predictors using different random weight initializations and training data orders. Then for any proposal, we can obtain the mean and standard deviation of these P predictions. More specifically, we train an ensemble of P predictive models, $\{f_p\}_{p=1}^P$, where $f_p : A \rightarrow \mathbb{R}$ with a pruning proposal g as input and the predicted reward as output. The mean prediction and its deviation are given by,

$$\hat{f}(g) = \frac{1}{P} \sum_{p=1}^{P} f_p(g), \text{ and } \hat{\sigma}(g) = \sqrt{\frac{\sum_{p=1}^{P} (f_p(g) - \hat{f}(g))^2}{P-1}}. \tag{3}$$

3) Selection with Acquisition Function: After training an ensemble of neural predictors, we can obtain the acquisition function value for proposals and select a small part of proposals with largest acquisition values. We choose upper confidence bound (UCB) [20] as the acquisition function,

$$\phi_{\text{UCB}}(g) = \hat{f}(g) + \beta \hat{\sigma}(g) \tag{4}$$

where the tradeoff parameter β is set to 0.5.

4) Evaluation with Magnitude Pruning: After selecting the pruning proposal from the pool, the evaluator uses magnitude based pruning framework [3] (with two steps including pruning and retraining) to perform the actual pruning and obtain its evaluation performance for the proposal. Note that it can evaluate the proposals in parallel. Besides, the speed measurement on a mobile device can be performed in parallel with the accuracy measurement.

IV. Experimental Results

A. Experiment Setup

We focus on 3D detection and employ the PointPillars [13] as starting point and test on KITTI dataset [21]. We use 40 GPUs for parallel training and pruning search and it takes about 6 days to find the best pruning proposal in each experiment. In Eq. (1), we set α to 0.01 and the mobile inference time is measured in milliseconds. The pool size C is set to 50 and the Bayesian batch size B is set to 10. We test the speed performance on the mobile GPU (Qualcomm Adreno 640) of a Samsung Galaxy S20 smartphone.

TABLE II
COMPARISON OF VARIOUS PRUNING METHODS FOR POINTPILLARS

Methods (grid size)	Para. #	Comp. # (MACs)	Speed (ms)	Car 3D detection Easy	Moderate	Hard
PointPillars (0.16)	5.8M	60G	553	85.16	74.39	69.42
Filter [22] (0.16)	1.1M	10.8G	178	80.63	67.51	65.28
Pattern [8] (0.16)	1.1M	10.7G	225	83.64	74.30	68.42
Block [10] (0.16)	1.1M	10.7G	268	82.86	75.43	69.71
Ours (0.16)	1.1M	10.7G	193	**85.52**	**76.69**	**70.10**
PointPillars (0.24)	5.4M	28G	253	84.24	75.28	68.46
Filter [22] (0.24)	0.8M	4.0G	82	81.36	68.06	65.77
Pattern [8] (0.24)	0.8M	3.9G	116	82.16	73.93	68.25
Block [10] (0.24)	0.8M	4.0G	140	83.69	74.09	68.06
Ours (0.24)	0.8M	3.9G	98	**85.38**	**75.72**	**68.53**

B. *Performance on 3D Object Detection*

As shown in Tab. II and Fig. 2, we compare the performance of the original unpruned PointPillars model and the model derived by our method and other pruning methods with different grid sizes (0.16m and 0.24m). We set the threshold of the real-time requirements to 200ms for 0.16m and 100ms for 0.24m. For the grid size, as large grid size leads to small pseudo-image input size for the model, the 0.24m grid size has a smaller parameter and computation numbers, and a faster inference speed on mobile GPUs, compared with 0.16m.

For the same grid size, compared with the original unpruned PointPillars model, we observe that our method can significantly reduce the number of parameters and computations, achieving state-of-the-art detection performance while satisfying the real-time requirement. The accuracy of our method is even higher than the unpruned model, demonstrating that the unpruned model may suffer from the over-fitting problem and removing the redundancy can help with its accuracy.

We also compare with other pruning methods for the same grid size. For other pruning methods, the same pruning scheme is applied to all layers and the pruning rate is set to the same with the overall pruning ratio of our pruned model (80% for grid size 0.16m and 86% for 0.24m). As observed, the proposed method achieves the best detection performance with highest accuracy compared with other methods with the same pruning scheme for every layer, demonstrating the advantages of different pruning scheme for different layers. We notice that although filter pruning can be faster than our method, it suffers from an obvious degradation on the detection performance.

For the speed, we notice that for grid size 0.24m, the proposed method only needs 98ms to process one LiDAR image on mobile devices with the highest accuracy, demonstrating its superior performance to achieve (close-to) real-time inference on mobile with state-of-the-art detection performance.

V. CONCLUSION

We propose pruning search to flexibly configure the pruning scheme and rate for each layer in the model with real-time inference requirement. Our experiments demonstrate that the proposed method achieves (close-to) real-time (98ms) 3D object detection based on PointPillars, on an off-the-shelf mobile phone with minor (or no) accuracy loss.

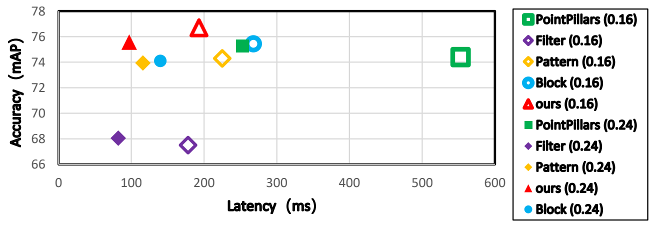

Fig. 2. Comparison with other methods

VI. ACKNOWLEDGEMENTS

This project is partly supported by National Science Foundation CNS-1929300, CNS-1739748, and CNS-1909172, Army Research Office/Army Research Laboratory (ARO) W911NF-20-1-0167 (YIP) to Northeastern University, a grant from Semiconductor Research Corporation (SRC), and Jeffress Trust Awards in Interdisciplinary Research. Any opinions, findings, and conclusions 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] W. Wen, C. Wu *et al.*, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016, pp. 2074–2082.
[2] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *NeurIPS*, 2016, pp. 1379–1387.
[3] S. Han, J. Pool *et al.*, "Learning both weights and connections for efficient neural network," in *NeurIPS*, 2015, pp. 1135–1143.
[4] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.
[5] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017, pp. 1389–1397.
[6] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICCV*, 2017, pp. 5058–5066.
[7] N. Liu, X. Ma *et al.*, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *AAAI*, 2020.
[8] X. Ma *et al.*, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," in *AAAI*, 2020.
[9] W. Niu *et al.*, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," *arXiv:2001.00138*, 2020.
[10] P. Dong, S. Wang *et al.*, "Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition," *arXiv:2002.11474*, 2020.
[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.
[12] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.
[13] A. H. Lang, S. Vora *et al.*, "Pointpillars: Fast encoders for object detection from point clouds," in *CVPR*, 2019, pp. 12 697–12 705.
[14] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.
[15] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1711–1719.
[16] H. Mao, S. Han *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv:1705.08922*, 2017.
[17] T. Zhang, S. Ye *et al.*, "Systematic weight pruning of dnns using alternating direction method of multipliers," *ECCV*, 2018.
[18] Z. Zhuang, M. Tan *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *NeurIPS*, 2018, pp. 875–886.
[19] Y. Chen, A. Huang *et al.*, "Bayesian optimization in alphago," *arXiv:1812.06855*, 2018.
[20] N. Srinivas, A. Krause *et al.*, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *ICML*, 2010.
[21] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.
[22] Y. He, P. Liu *et al.*, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019.