Review Article
Volume 10:8, 2021

International Journal of
Sensor Networks and Data Communications

ISSN: 2090-4886 Open Access

A Compiler-Aware Framework of Network Pruning and Architecture Search for Mobile Acceleration

Zhengang Li*

Department of Electrical and Computer Engineering, Northeastern University, Boston, United States

Abstract

With the increasing demand to efficiently deploy DNNs on mobile edge devices, it becomes much more important to reduce unnecessary computation and increase the execution speed. Prior methods towards this goal, including model compression and network architecture search (NAS), are largely performed independently and do not fully consider compiler-level optimization which is a must-do for mobile acceleration. In this work, we propose NPAS, a compiler-aware unified network pruning and architecture search and the corresponding comprehensive compiler optimizations supporting different DNNs and different pruning schemes, which bridge the gap of weight pruning and NAS. Our framework achieves 6.7 ms, 5.9 ms, and 3.9 ms ImageNet inference times with 78%, 75% (MobileNet-V3 level), and 71% (MobileNet-V2 level) Top-1 accuracy respectively on an off-the-shelf mobile phone, consistently outperforming prior work.

Keywords

Artificial intelligence ● Neural architecture search ● Network pruning and architecture search ● Deep neural network

Introduction

The growing popularity of mobile AI applications and the demand for real-time Deep Neural Network (DNN) executions raise significant challenges for DNN accelerations. However, the ever-growing size of DNN models causes intensive computation and memory cost, which impedes the deployment on resource limited mobile devices.

DNN weight pruning has been proved as an effective model compression technique that can remove redundant weights of the DNN models, thereby reducing storage and computation costs simultaneously [1,2]. Existing work mainly focus on unstructured pruning scheme where arbitrary weight can be removed as shown in and (coarse-grained) structured pruning scheme to eliminate whole filters/channels as shown in Figures 1a and 1b [1,3]. The former results in high accuracy but limited hardware parallelism (and acceleration), while the latter is the opposite. Recent work propose to prune the weights in a more fine-grained manner, which can be classified into block-based and pattern-based pruning as shown in Figures 1c and 1d [4,5]. This kind of semi-structured pruning preserves higher accuracy while also provides significant speedup with the assist of compiler-level code generation techniques.

***Address for Correspondence:** Zhengang Li, Department of Electrical and Computer Engineering, Northeastern University, Boston, United States; E-mail: li.zhen@northeastern.edu*

Received: August 04, 2021; **Accepted:** August 18, 2021; **Published:** August 25, 2021

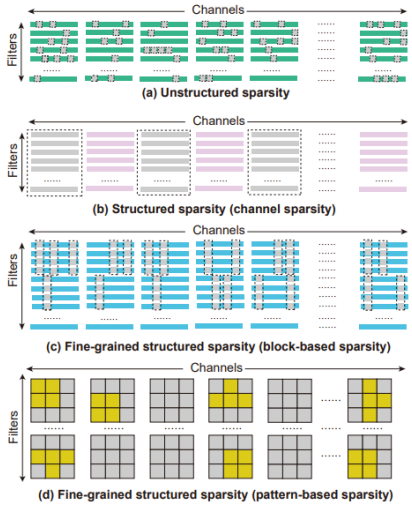

Figure 1. Different weight pruning schemes for CONV and FC layers using 4D tensor and 2D matrix representation. (a) Unstructured sparsity, (b) Structured sparsity (channel sparsity), (c) Fine-grained structured sparsity (block-based sparsity), (d) Fine-grained structured sparsity (pattern-based sparsity).

Literature Review

Another active research area is the Neural Architecture Search (NAS), which designs more efficient DNN architectures using automatic searching algorithms [6]. Efficient Net and MobileNetV3 is representative lightweight net- works obtained by using NAS approaches [7,8]. Hardware aware NAS has also been investigated targeting acceleration on actual hardware platforms [9,10]. Recently, compiler-assisted DNN inference frame- works have drawn broad attention from both industry and academia [11,12]. TensorFlow- Lite (TFLite) (Ten), Alibaba Mobile Neural Network (MNN) (Ali), and TVM are representative state-of-the-art frameworks that support DNN inference on mobile devices [13-15]. Recent work Pat DNN employs a set of compiler-based optimizations to support specific pattern-based sparse DNN models to accelerate the end-to-end inference on mobile devices [5]. However, it still lacks the support for a layer-wise sparse model with various pruning schemes, which significantly limits the versatility of such framework.

Pruning a DNN model for real-time AI applications on mobile devices is a complex task because different types of layers may prefer different types of pruning schemes. At the same time, different layers may show different

sensitivities to the pruning ratio. Moreover, even under the similar pruning ratio, different pruning schemes also perform different acceleration rates due to computing parallelism. Thus, we bridge the weight pruning technique and NAS methods and propose a reinforcement learning (RL)-based network pruning and architecture search framework to automatically search the best-suited pruning configurations such as per- layer pruning scheme and pruning ratio. Moreover, we propose multiple compiler optimizations to enable fast code generation and support inference acceleration with per-layer pruning schemes and ratios. We incorporate the compiler optimized model inference latency measured on the target mobile device as a reward in the searching process, making our framework compiler-aware. Our key contributions include:

• We bridge the gap between network pruning and NAS. We develop a compiler-aware framework of network pruning and architecture search, maximizing accuracy while satisfying inference latency constraint.

• We propose comprehensive compiler optimizations supporting different pruning schemes and sparse model inference with per-layer pruning schemes.

• We design a systematic search acceleration strategy, integrating pre-trained starting points, fast accuracy and latency evaluations, and Bayesian optimization.

• Our NPAS framework achieves by far the best mobile acceleration: 6.7ms, 5.9ms, and 3.9ms ImageNet inference times with 78%, 75%, and 71% Top-1 accuracy, respectively, on an off-the-shelf mobile phone.

Proposed Unified Network Pruning and Architecture Search (NPAS) Framework

Overview of NPAS framework

It shows the proposed NPAS framework Figure 2. To take advantage of recent NAS results and accelerate the NPAS process, we start from a pre-trained DNN model, and go through three phases as shown in the figure.

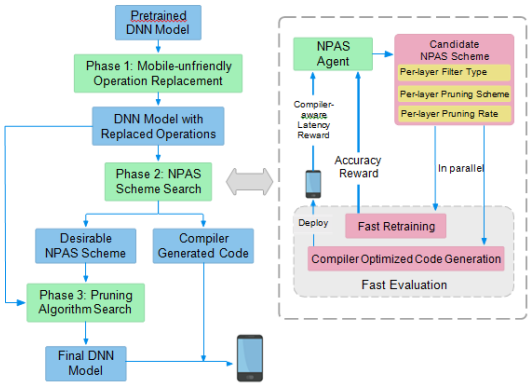

Figure 2. Overview of the proposed NPAS framework.

Phase 1 replacement of mobile-unfriendly operations: Certain operators are inefficient to execute on mobile devices (mobile CPU and GPU). For instance, certain activation functions, such as sigmoid, swish, require exponential computation, and can become latency bottleneck on mobile inference. These unfriendly operations will be replaced by mobile-friendly alternatives such as hard-sigmoid and hard-swish, with negligible effect on accuracy.

Phase 2 NPAS scheme search: This phase uses a RL- based NAS method to generate and evaluates candidate NPAS schemes, and finally chooses the best-suited one. The search space includes per-layer filter type, per-layer prunning scheme and per-layer pruning rate. To accelerate such search, we present a meta-modelling procedure based on RL with Bayesian Optimization (BO). A fast evaluation methods are developed, tailored to NPAS framework. Moreover, we incorporate the overall DNN latency constraint effectively in the reward function of NPAS scheme search, ensuring that such constraint can be satisfied at the search outcome. The overall DNN latency is actually measured on the target mobile CPU/GPU based on the candidate NPAS scheme currently under evaluation. We rely on actual measurement instead of per-layer latency modelling as many prior NAS work. This is because our advanced compiler optimizations incorporate a strong layer fusion beyond prior compiler work, which is critical for efficient implementation of super-deep networks, and will make per-layer latency modelling less accurate.

Phase 3 Pruning algorithm search: We search the most desirable pruning algorithm to perform actual pruning and retrain the remaining weights. The candidate pruning algorithms include magnitude-based ones, ADMM-based algorithm etc [16,17].

Fast evaluation methods

We develop and adopt multiple tailored acceleration strategies to facilitate fast evaluation in NPAS scheme search. To evaluate each generated candidate scheme during search, we use the one-shot magnitude pruning instead of using complex pruning algorithms. And we adopt early stop- ping strategy, which only retraining pruned model for a few epochs. Because we can distinguish the performance of a candidate NPAS scheme by comparing the relative accuracy to other NPAS schemes. Moreover, we overlap the compiler optimization process with the accuracy evaluation process to further accelerate the overall evaluation process. We use compiler code generation and actual on-device latency measurement because of

• Higher accuracy than per-layer latency modelling due to layer fusion mechanism, and

• The fast auto-tuning capability of compiler to different mobile devices.

Please note that the compiler code generation and latency measurement do not need the absolute weight values. Compiler code generation is much faster than DNN training (even a single epoch), and can be performed in parallel with accuracy evaluation (as accurate weight values are not needed). As a result, it will not incur extra time consumption to NPAS.

Discussion

Compiler design and optimizations

Another source of acceleration to achieve real-time inference on mobile devices is the compiler optimizations for generating efficient execution codes. We develop a comprehensive, compiler-based automatic code generation method with multiple optimizations.

Support for various pruning schemes: We design a domain specific language (DSL) to represent the DNN model, and a layer-wise representation (LR) is used to describe each DNN layer. This provide us the flexibility for supporting the layer-wise pruning scheme selection. We also design compact weight storage formats for different pruning schemes to improve the data locality.

Layer fusion mechanism: We incorporate a layer fusion technique to fuse the computation operators in computation graph and effectively reduce the inference latency. Our fusion based on two kinds of properties in the polynomial calculation: computation laws (i.e., associative property, commutative property, and distributive property) and data access patterns. As a result, we reduce not only the memory consumption of intermediate results, but also the number of operators.

Auto-tuning for different mobile CPU/GPU: To find the best-suited performance-critical tuning parameters, such as the data placement on GPU memory, matrix tiling sizes, loop unrolling factors, we use auto-

tuning approaches as other DNN inference frameworks like TVM. And we incorporate Genetic Algorithm to explore the best configuration automatically and efficiently.

Compiler-aware latency: The latency of a given candidate model is hard to be accurately estimated based on a layer-wise latency model when compiler optimizations are incorporated, especially with layer fusion and auto-tuning. Thus, during the search process, we use real-world compiler optimized latency measured on the real device instead of building a layer-wise latency model. Since the code generation time of our optimized compiler design is much shorter than the accuracy evaluation process, we overlap the code generation and latency measurement with the accuracy evaluation process; hence no extra time cost will be incurred.

Comparison with representative DNN inference acceleration frameworks on mobile device: To demonstrate the generality and the superiority of our compiler optimizations, we compared the inference latency of both dense model and sparse model with other representative DNN inference acceleration frameworks including TFLite, TVM, and MNN. And we show the results on widely used benchmark networks including VGG-16, ResNet-18 and MobileNet-V2. Tests are conducted on a Samsung Galaxy S10 smartphone with mobile CPU and mobile GPU respectively. As shown in, only based on our compiler optimization (without pruning), our results clearly outperforms the representative frameworks on both mobile CPU and mobile GPU (Table 1). By incorporating our network pruning (without causing accuracy loss), the inference latency is further reduced. The pruning rate for VGG-16, ResNet-18, and MobileNet-V2 is 8.2×, 5.3×, and 1.8×, respectively.

Results and evaluation

Experimental setup: We use the image classification task and ImageNet dataset to show the effectiveness of our framework, as in Figures 3 and 4. We compare our accuracy and latency results with representative DNN inference acceleration frameworks including MNN, PyTorch Mobile, and TFLite. The results are tested on a Samsung Galaxy S10 smartphone using mobile CPU (Qualcomm Kryo 485) or mobile GPU (Qualcomm Adreno 640). For Phase 1, we conduct a fast fine-tuning with 5 training epochs after replacing the mobile-unfriendly operations (only once for the entire NPAS process). In Phase 2, 40 Nvidia Titan RTX GPUs are used to conduct the fast accuracy evaluation for candidate NPAS schemes concurrently.

Since we start from a well-trained model, we retrain 2 epochs for each candidate one-shot pruned model for fast evaluation. For each candidate model, we measure 100 runs of inference on target mobile devices and use the average value as end- to-end latency. Thanks to our fast evaluation and BO, using EfficientNet- B0 as starting point, the overall searching time is 15 days, where Phase 1 only takes 5 epochs, and Phase 3 takes 1.5 days.

Evaluation results

First, our compiler optimizations can effectively speed up inference by up to 46% and 141% (on MobileNet-V3) without incorporating NPAS compared to the currently best frame- work MNN on mobile CPU and GPU, respectively.

With the highest accuracy (78.2% Top-1), the end-to-end inference time of NPAS solution (385M MACs) is only 11.8ms and 6.7ms on mobile CPU and GPU, respectively. With MobileNet-V3 level accuracy (75% Top-1); our inference time (201M MACs) is 9.8ms and 5.9ms. With MobileNet-V2 level accuracy (71% Top-1); the inference time of NPAS solution (147M MACs) is 6.9ms and 3.9ms. To the best of our knowledge, this is never accomplished by any existing NAS or weight pruning work. Detailed results can be found in Table 2 [18-21].

Framework	VGG-16	ResNet-18	MobileNet-V2
TF lite	429 / 307	108 / 49.9	55.2 / 24.3
TVM	251 / 221	61.5 / 37.6	23.1 / 20.5
MNN	239 / 141	52.4 / 23.7	18.6 / 14.5
Ours (dense)	204 / 103	41.1 / 19.8	17.4 / 9.3
Ours (sparse)	37.3 / 18.1	20.6 / 9.7	9.2 / 4.3

Table 1. Mobile CPU/GPU Inference latency (ms) comparison with MNN, TVM, and TF Lite using dense (unpruned) models.

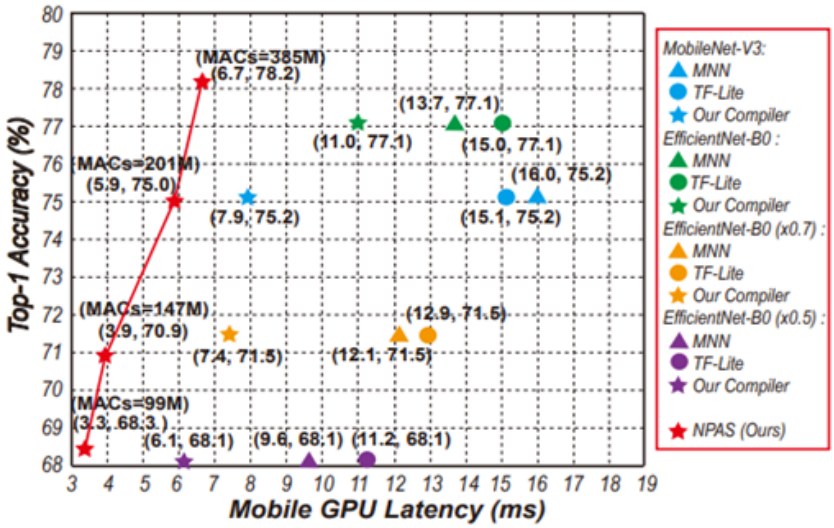

Figure 3. Accuracy vs. latency comparison on mobile GPU.

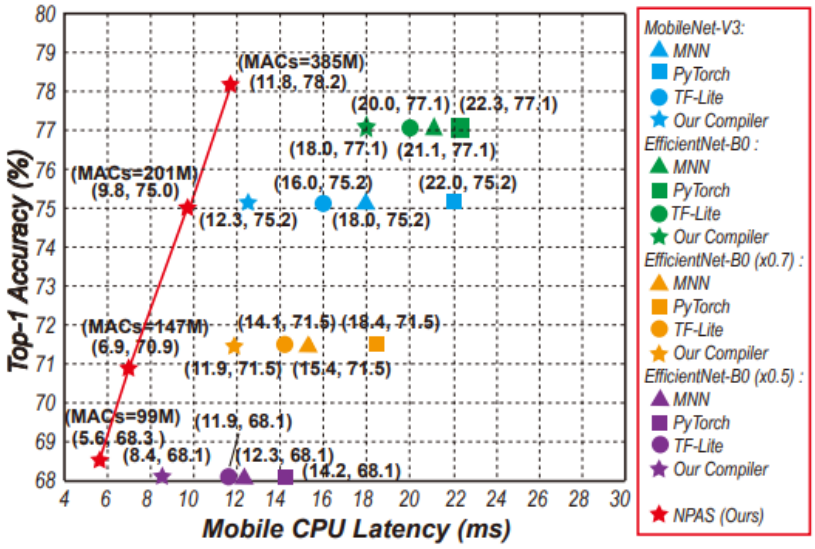

Figure 4. Accuracy vs. latency comparison on mobile CPU.

MACs	Acc. top-1	Latency (ms)	CPU/GPU	Device
MobileNet-V1	575M	70.6	- / -	-
MobileNet-V2	300M	72	- / -	-
MobileNet-V3	227M	75.2	- / -	-
NAS-Net-A	564M	74	183 / NA	Google Pixel 1
AmoebaNet-A	555M	74.5	190 / NA	Google Pixel 1
MnasNet-A1	312M	75.2	78 / NA	Google Pixel 1
ProxylessNas-R	NA	74.6	78 / NA	Google Pixel 1
NPAS (ours)	385M	78.2	11.8 / 6.7	Galaxy S10
NPAS (ours)	201M	75	9.8 / 5.9	Galaxy S10
NPAS (ours)	147M	70.9	6.9 / 3.9	Galaxy S10
NPAS (ours)	98M	68.3	5.6 / 3.3	Galaxy S10

Table 2. Comparison results of NPAS and representative lightweight networks: MobileNet-V1 [18], MobileNet-V2 [19], MobileNet-V3 [8], NAS-Net-A [20], AmoebaNet-A [21], MnasNet-A1 [10], ProxylessNas-R [9].

Conclusion

In this work, we propose a fine-grained structured pruning applicable to various DNN layers, and a compiler automatic code generation framework supporting different DNNs and different pruning schemes, which bridge the gap of model compression and NAS. We further propose NPAS, a compiler-aware unified network pruning and architecture search, and several techniques are used to accelerate the searching process.

Conflict of Interest

Author has nothing to disclose.

References

1. Han, Song, Jeff Pool, John Tran and William J. Dally. "Learning both Weights and Connections for Efficient Neural Networks." *Proc Int Conf Neural Inf Process Syst* 1 (2015): 1135-1143.
2. He, Yihui, Ji Lin, Zhijian Liu and Hanrui Wang, et al. "AMC: AutoML for Model Compression and Acceleration on Mobile Devices." *Proc Eur Conf Comput Vision* 1 (2018): 784-800.
3. Zhuang, Zhuangwei, Mingkui Tan, Bohan Zhuang and Jing Liu, et al. "Discrimination-Aware Channel Pruning for Deep Neural Networks." *Proc Int Conf Neural Inf Process Sys* 1 (2018): 883-894.
4. Dong, Peiyan, Siyue Wang, Wei Niu and Chengming Zhang, et al. "RTMobile: Beyond Real-Time Mobile Acceleration of RNNs for Speech Recognition." *IEEE Des Autom Conf* 1 (2020): 11474.
5. Niu, Wei, Xiaolong Ma, Sheng Lin and Shihao Wang, et al. "PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-Based Weight Pruning." *Proc Int Conf Archit Support Program Lang Operating Sys* 1 (2020): 907-922.
6. Zoph, Barret and Quoc V Le. "Neural Architecture Search with Reinforcement Learning." *Int Conf Learn Representations* 1 (2017): 1-16.
7. Tan, Mingxing and Quoc V Le. "Efficient Net: Rethinking Model Scaling for Convolutional Neural Networks." *Int Conf Mach Learn* 1 (2019): 11946.
8. Howard, Andrew, Ruoming Pang, Hartwig Adam and Quoc Le, et al. "Searching *for MobileNetV3." IEEE Int Conf Comput Vis* 1 (2019): 1314-1324.
9. Cai, Han, Ligeng Zhu and Song Han. "Proxyless NAS: Direct Neural Architecture Search on Target Task and Hardware." *Mach Learn* 1 (2018): 1-13.
10. Tan, Mingxing, Bo Chen, Ruoming Pang and Vijay Vasudevan, et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile." *IEEE Conf Comput Vis Pattern Recognit* 1 (2019): 2815-2823.
11. Lane, D Nicholas, Petko Georgiev and Lorena Qendro. "Deepear: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning." *Proc ACM Int Joint Conf on Pervasive Ubiquitous Comput* 1 (2015): 283-294.
12. Xu, Mengwei, Mengze Zhu, Yunxin Liu and Felix Xiaozhu Lin, et al. "Deep Cache: Principled Cache for Mobile Deep Vision." *Proc Annu Int Conf Mobile Comput Networking* 1 (2018): 129-144.

13. Mobile Neural Network. Alibaba.

14. Deploy Machine Learning Models on Mobile And IoT Devices. TensorFlow.

15. Chen, Tianqi, Thierry Moreau, Ziheng Jiang and Lianmin Zheng, et al. "TVM: An Automated End-To-End Optimizing Compiler for Deep Learning." *Proc USENIX Conf Oper Sys Des Implement* 1 (2018): 578-594.

16. Han, Song, Huizi Mao and William J. Dally. "*Deep Compression: Compressing* Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." *Int Conf Learn Representations* 1 (2016): 1-10.

17. Zhang, Tianyun, Shaokai Ye, Kaiqi Zhang and Yanzhi Wang, et al. "A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers." *Eur Conf Comput Vis* 11212 (2018): 191-207.

18. Howard, Andrew, Menglong Zhu, Bo Chen and Dmitry Kalenichenko, et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." *Comput Vis Pattern Recognit* 1 (2017): 1-12.

19. Real, Esteban, Alok Aggarwal, Yanping Huang, and Quoc V Le. "Regularized Evolution for Image Classifier Architecture Search." *Proc AAAI Conf Artif Intell* 33 (2019): 4780-4789.

20. Sandler, Mark, Andrew Howard, Menglong Zhu and Andrey Zhmoginov, et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." *Proc IEEE Conf Comput Vis Pattern Recognit* 1 (2018): 4510- 4520.

21. Zoph, Barret, Vijay Vasudevan, Jonathon Shlens and Quoc V Le. "Learning Transferable Architectures for Scalable Image Recognition." *Proc IEEE Conf Comput Vis Pattern Recognit* 1 (2018): 8697-8710.

How to cite this article: Li, Zhengang."A Compiler-Aware Framework of Network Pruning and Architecture Search for Mobile Acceleration." *J Sens Netw Data Commun* 10 (2021): 138.