PareCO: Pareto-aware Channel Optimization for Slimmable Neural Networks

Ting-Wu Chin TINGWUC@CMU.EDU
Department of Electrical and Computer Engineering
Carnegie Mellon University

Ari S. Morcos ARIMORCOS@FB.COM
Facebook AI Research

Diana Marculescu DIANAM@UTEXAS.EDU
Department of Electrical and Computer Engineering
The University of Texas at Austin & Carnegie Mellon University

Abstract

Slimmable neural networks have been proposed recently for resource-constrained settings such as mobile devices as they provide a flexible trade-off front between prediction error and computational cost (such as the number of floating-point operations or FLOPs) with the same storage cost as a single model. However, current slimmable neural networks use a single width-multiplier for all the layers to arrive at sub-networks with different performance profiles, which neglects that different layers affect the network's prediction accuracy differently and have different FLOP requirements. We formulate the problem of optimizing slimmable networks from a multi-objective optimization lens, which leads to a novel algorithm for optimizing both the shared weights and the width-multipliers for the sub-networks. While slimmable neural networks introduce the possibility of only maintaining a single model instead of many, our results make it more realistic to do so by improving their performance.

Keywords: Model Compression, Multi-objective Bayesian Optimization

1. Introduction

Slimmable neural networks have been proposed with the promise of enabling multiple neural networks with different trade-offs between prediction error and the number of floating-point operations (FLOPs), *all at the storage cost of only a single neural network* (Yu et al. [2019]). Such neural networks are useful for applications on mobile and other resource-constrained devices. As an example, the ability to deploy multiple versions of the same neural network would alleviate the maintenance costs for applications which support a number of different mobile devices with different memory and storage constraints, as only one model needs to be maintained. Similarly, one can deploy a single model which is configurable at run-time to dynamically cope with different latency or accuracy requirements. For example, users may care more about power efficiency when the battery of their devices is running low while the accuracy of the application may be more important otherwise.

A slimmable neural network is trained by simultaneously training networks with different widths (or channel counts) using a single set of shared weights. The width of a child network is specified by a real number between 0 and 1, which is known as the "width-multiplier" (Howard et al. [2017]). Such a parameter specifies how many channels per layer to use proportional to the full network. For example, a width-multiplier of 0.35× represents a network that has channel counts that are 35% of the full network for all the layers. While specifying child networks using a single width-multiplier for all the layers has shown empirical success (Yu and Huang [2019b]; Yu et al. [2019]), such a specification neglects that different layers affect the network's output differently Zhang et al. [2019] and have different FLOP requirements (Gordon et al. [2018]), which may lead to sub-optimal results. To support heterogeneous width-multipliers across layers, we take a multi-objective optimization viewpoint, aiming to jointly optimize the width-multipliers for different layers and the shared weights in a slimmable neural network. A schematic view of the difference between the proposed and the conventional slimmable networks is shown in Figure 2 in Appendix.

2. Methodology

2.1 Problem formulation

Intuitively, our goal is to optimize the shared weights to maximize the area under the best trade-off curve between the accuracy and theoretical speedup obtained by optimizing network's widths. Since accuracy is not differentiable w.r.t. the shared weights, we switch objectives from accuracy and theoretical speedup to cross-entropy loss and FLOPs, respectively. In this setting, the objective becomes to *minimize* the area under curve. To arrive at such an objective, we start by defining the notion of optimality in *minimizing* multiple objectives (such as the cross-entropy loss and FLOPs).

Definition 1 (Pareto frontier) *Let $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \dots, f_K(\boldsymbol{x}))$ be a vector of responses from K different objectives. Define vector inequality $\boldsymbol{x} < \boldsymbol{y}$ as $x_i \leq y_i \; \forall \; i \in [K]$ with at least one inequality being strict. We call a set of points $\mathcal{P}$ a Pareto frontier if $\boldsymbol{f}(\boldsymbol{x}) < \boldsymbol{f}(\boldsymbol{y})$, for any $\boldsymbol{x} \in \mathcal{P}$ and $\boldsymbol{y} \notin \mathcal{P}$.*

With this definition, we essentially want the loss for the shared weights to be the area under the curve formed by the Pareto frontier. To do so, we need an actionable way to obtain the Pareto frontier and we make use of the following Lemma:

Lemma 2 (Augmented Tchebyshev Scalarization) *Define a scalarization of K objectives as*

$$\mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{x}) = \max_{i \in [K]} \lambda_i (f_i(\boldsymbol{x}) - \bar{f}_i) + \beta \sum_{i \in [K]} \lambda_i f_i(\boldsymbol{x}), \tag{1}$$

where $\bar{f}_i$ is a baseline constant such that $(f_i(\boldsymbol{x}) - \bar{f}_i) \geq 0 \; \forall \; \boldsymbol{x}$, and $\beta > 0$, the Pareto frontier can be specified via $\mathcal{P} = \{\arg\min_{\boldsymbol{x}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{x}) \; \forall \boldsymbol{\lambda} \in \Delta^{K-1}\}$ where Δ^{K-1} is a K-1 simplex. (Nakayama et al. [2009], Section 1.3.3)

With Lemma 2, one can obtain the Pareto frontier by solving multiple augmented Tchebyshev scalarized optimization problems with different $\boldsymbol{\lambda}$s. A $\boldsymbol{\lambda}$ vector can be interpreted as a weighting on the objectives, which is used to summarize multiple objectives into a single scalar. For instance, consider the case in which the cross-entropy loss and FLOPs are the two objectives of interest. If taking $\lambda_{\text{CE}} \to 1$ and $\lambda_{\text{FLOPs}} \to 0$, the scalarized objective is then dominated by the cross-entropy loss and we are effectively seeking width configurations that minimize the cross entropy loss. One can further summarize the area under the Pareto curve to a scalar for optimization by taking an expectation over $\boldsymbol{\lambda}$, which takes into account the loss incurs by multiple weightings. Since the shared weights $\boldsymbol{\theta}$ only affect the cross-entropy loss but not FLOPs, minimizing the cross-entropy loss induced by the widths on the Pareto frontier effectively minimizes the area under curve. We can formally define our problem of interest:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}} \mathbb{E}_{\boldsymbol{\lambda}\sim\mathcal{L}} [f_{CE}(\boldsymbol{\theta}, \boldsymbol{\alpha}_{\boldsymbol{\lambda}}, \boldsymbol{x}, y)] \\ &\text{s.t.} \quad \boldsymbol{\alpha}_{\boldsymbol{\lambda}} = \min_{\boldsymbol{\alpha}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{\alpha}; \boldsymbol{\theta}, \boldsymbol{x}, y) \end{aligned} \tag{2}$$

where $\boldsymbol{\theta}$ denotes the network weights we would like to optimize, $\boldsymbol{\alpha}$ denotes the network widths, $\mathcal{L}$ denotes the distribution that governs the regions of interest on the Pareto front and it has support over Δ^{K-1}. $\mathcal{D}$ denotes the distribution of the training data, and $\boldsymbol{x}$ and y are the training input and label. The expectation over $\boldsymbol{\lambda}$ summarizes the area under the trade-off curve. Note that $\mathcal{T}_{\boldsymbol{\lambda}}(\cdot)$ is implicitly conditioned on $\boldsymbol{\theta}$, $\boldsymbol{x}$, and y due to the cross-entropy loss.

While equation (2) precisely defines our goal, solving the constraint can be intractable since the function is usually highly non-convex with respect to $\boldsymbol{\alpha}$ and does not have analytical gradient information that admits first-order optimization algorithms. To cope with these challenges, we adopt multi-objective Bayesian Optimization with randomize scalarization (MOBO-RS) Paria et al. [2019] to approximate the minimization in the constraint.

2.2 PareCO: Pareto-aware channel optimization

The proposed framework has three steps: (1) sample $\boldsymbol{\lambda^{(m)}}$, (2) solve for the corresponding Pareto-optimal width configurations $\widehat{\boldsymbol{\alpha^{(m)}}}$ via MOBO-RS, and (3) update weights by running forward and backward passes using these network widths. We call a *full iteration* to be one iteration of the three steps. We avoid superscripts when possible in the sequel.

Pareto-aware sampling To sample diverse solutions on the Pareto curve, we make use of the width configurations obtained so far across full iterations $\mathcal{H}$ to obtain an approximate Pareto frontier. Specifically, the approximate Pareto frontier $\mathcal{N} \subset \mathcal{H}$ is defined such that $\boldsymbol{f(x)} < \boldsymbol{f(y)}\ \forall\ \boldsymbol{x} \in \mathcal{N}, \boldsymbol{y} \notin \mathcal{N}$. Based on $\mathcal{N}$, we would like to quantify the level of under-exploration for the Pareto curve. For example, in the Pareto frontier defined by cross-entropy loss and FLOPs, the level of under-exploration can be characterized by the area between two consecutive points for both the cross-entropy loss and FLOPs. Formally,

$$A_i \doteq \left(f_{\text{FLOPs}}(\boldsymbol{z_{i+1}}) - f_{\text{FLOPs}}(\boldsymbol{z_i})\right)\left(f_{\text{CE}}(\boldsymbol{z_i}) - f_{\text{CE}}(\boldsymbol{z_{i+1}})\right), \tag{3}$$

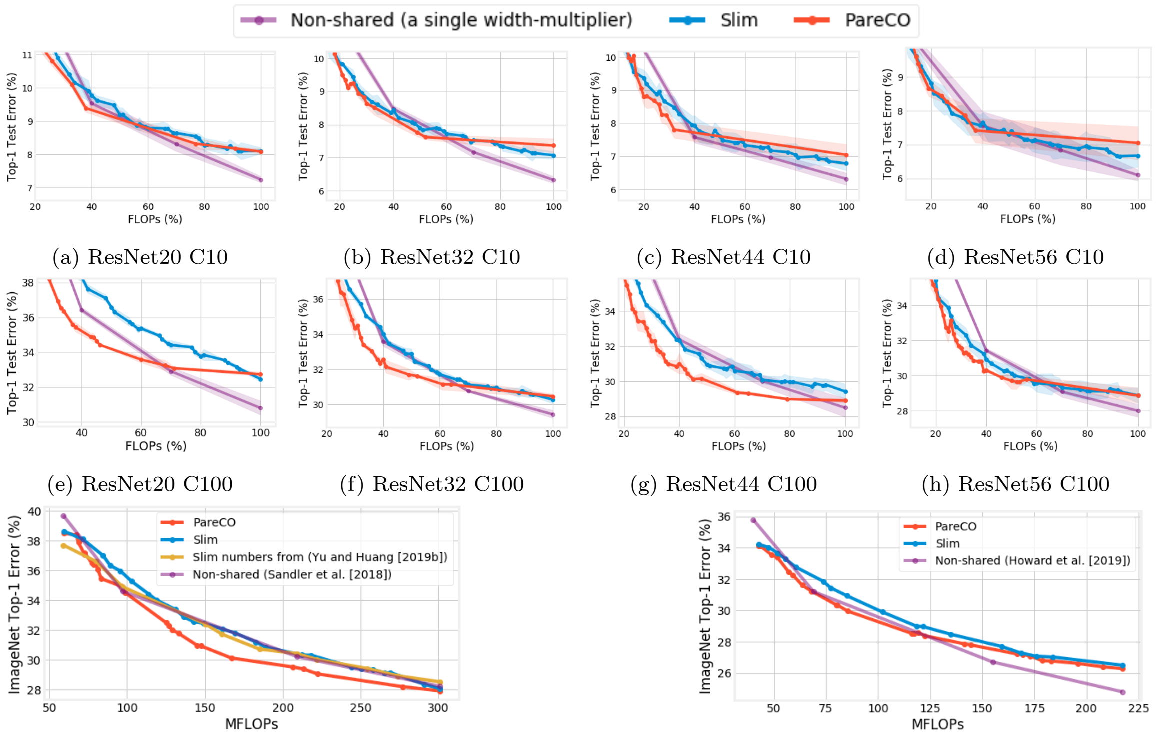
where $\boldsymbol{z} \in \mathcal{N}$ is ordered solutions sorted in ascending order according to $f_{\text{FLOPs}}(\cdot)$.

Using A_i to quantify under-exploration, our strategy to sample $\boldsymbol{\lambda}$ involves first sampling a target function value $\tilde{f}_{\text{FLOPs}}$ such that $\mathbb{P}(\tilde{f}_{\text{FLOPs}} \in [f_{\text{FLOPs}}(\boldsymbol{z_i}), f_{\text{FLOPs}}(\boldsymbol{z_{i+1}})]) \propto A_i$. Then, we solve the scalarized acquisition function with an initial $\boldsymbol{\lambda} = \{\lambda_{\text{FLOPs}}, \lambda_{\text{CE}}\}$ to obtain $\widehat{\boldsymbol{\alpha}}$. If $f_{\text{FLOPs}}(\widehat{\boldsymbol{\alpha}})$ is larger than $\tilde{f}_{\text{FLOPs}}$, λ_{FLOPs} is increased; otherwise λ_{FLOPs} is decreased. Binary search is repeated until $\frac{|f_{\text{FLOPs}}(\widehat{\boldsymbol{\alpha}}) - \tilde{f}_{\text{FLOPs}}|}{\text{FullModelFLOPs}} \leq \epsilon$ or until a pre-defined number of iterations is met.

One-step MOBO-RS Since MOBO-RS itself is a sequential optimization process, running many iterations of MOBO-RS for each full iteration and $\boldsymbol{\lambda}$ could be costly. To reduce this cost, our intuition is that the cross-entropy loss has high variance throughout the early phase of training, which makes the precise minimizer $\widehat{\boldsymbol{\alpha}}$ less useful. As a result, we propose to perform one-step optimization in each MOBO-RS by sharing the queries visited. That is, the output of MOBO-RS at full iteration t is obtained by optimizing the posterior formed by the minimizers obtained in earlier full iterations $\mathcal{H} = \{\widehat{\boldsymbol{\alpha_1}}, \dots, \widehat{\boldsymbol{\alpha_{t-1}}}\}$. Note that to make use of the historical data, $|\mathcal{H}|$ forward passes are needed to obtain the cross-entropy losses at each $\widehat{\boldsymbol{\alpha}}$ with the latest $\boldsymbol{\theta}, \boldsymbol{x}$, and y for building the Gaussian Process. This approximation allocates less information in earlier full iterations and assumes that the underlying function $f_{\text{CE}}(\boldsymbol{\alpha}; \boldsymbol{\theta}, \boldsymbol{x}, y)$ would not change drastically across each full iteration.

Increasing the number of gradient descent iterations This expensive optimization procedure (*i.e.*, MOBO-RS) is called every full iteration. To reduce the overall training time, we can perform multiple gradient updates per full iteration of MOBO-RS, under the assumption that a slightly stale $\mathcal{H}$ will not fundamentally change learning. We term this hyperparameter n and evaluate its impact in Appendix F.

PareCO Based on this preamble, we propose PareCO, which is detailed in Algorithm 1 in Appendix B. In short, PareCO has three steps: (1) build surrogate functions (*i.e.*, GPs) and acquisition functions (*i.e.*, UCBs) using historical data $\mathcal{H}$ and their function responses, (2) sample M $\boldsymbol{\lambda}$s and



(a) ResNet20 C10 (b) ResNet32 C10 (c) ResNet44 C10 (d) ResNet56 C10

(e) ResNet20 C100 (f) ResNet32 C100 (g) ResNet44 C100 (h) ResNet56 C100

(i) MobileNetV2 ImageNet (j) MobileNetV3 ImageNet

Figure 1: Comparisons among PareCO and Slim. C10 and C100 denote CIFAR-10/100. For the CIFAR dataset, we perform three trials for each method and plot the mean and standard deviation. PareCO is better or comparable to Slim.

solve for the corresponding widths (*i.e.*, $\widehat{\boldsymbol{\alpha}}$) via pareto-aware sampling, and (3) perform n gradient descent steps using the solved widths.

3. Experiments

We are interested in the following question: *Do PareCO-optimized models (PareCO) outperform conventional slimmable networks (Slim) (Yu and Huang [2019b])?* To answer this question, we compare both algorithms using the exact same code base and training hyperparameters. We consider ResNets with various depths targeting CIFAR-10 and CIFAR-100. As shown in Figure 1, PareCO improves Slim in most cases for CIFAR-100 while PareCO performs similarly to or slightly better than Slim for CIFAR-10. Interestingly, we find that when the network is relatively more over-parameterized, the two perform more similarly, as it can be seen in Figure 1d. This is plausible since when a network is more over-parameterized, there are many solutions to the optimization problem and it is easier to find solutions with the constraints imposed by weight sharing. In contrast, when the network is relatively less over-parameterized, compromises have to be made due to the constraints imposed by weight sharing. In such scenarios, PareCO outperforms Slim significantly, as it can be seen in Figure 1e. We conjecture that this is because PareCO introduces a new optimization variable (width-multipliers), which allows better compromises to be attained. We further conduct the same experiments on ImageNet using MobileNetV2 (Sandler et al. [2018]) and MobileNetV3 (Howard et al. [2019]). As shown in Figure 1i and Figure 1j, we observe the similar trend that PareCO outperforms Slim. Compared to non-shared models using a single width-multiplier, PareCO manages to perform better in low-FLOPs regimes with optimized widths.

4. Conclusion

In this work, we propose to tackle the problem of training slimmable networks via a multi-objective optimization lens, which provides a novel and principled framework for optimizing slimmable networks. With this formulation, we propose a novel training algorithm, PareCO, which trains slimmable neural networks by jointly learning both channel configurations and the shared weights. Our results highlight the potential of optimizing the channel counts for different layers jointly with the weights and demonstrate the power of such techniques for slimmable networks.

References

Yonathan Aflalo, Asaf Noy, Ming Lin, Itamar Friedman, and Lihi Zelnik. Knapsack pruning with inner distillation. *arXiv preprint arXiv:2002.08258*, 2020.

Maximilian Balandat, Brian Karrer, Daniel R Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*, 2019.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/bender18a.html`.

Maxim Berman, Leonid Pishchulin, Ning Xu, Gérard Medioni, et al. Aows: Adaptive and optimal network width search with latency constraints. *Proceedings IEEE CVPR*, 2020.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 527–536. JMLR. org, 2017.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HylxE1HKwS`.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018a.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018b.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Yiğitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-Deep Networks: Understanding and mitigating network overthinking. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*, Long Beach, CA, Jun 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1891–1900, 2019a.

Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019a.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019b. URL `https://openreview.net/forum?id=rJlnB3C5Ym`.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017a.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through *l*_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017b.

Xingchen Ma, Amal Rannen Triki, Maxim Berman, Christos Sagonas, Jacques Cali, and Matthew B Blaschko. A bayesian optimization framework for neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10274–10283, 2019.

Bertil Matérn. *Spatial variation*, volume 36. Springer Science & Business Media, 2013.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

Hirotaka Nakayama, Yeboon Yun, and Min Yoon. *Sequential approximate multiobjective optimization using computational intelligence*. Springer Science & Business Media, 2009.

Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible framework for multi-objective bayesian optimization using random scalarizations. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 267. AUAI Press, 2019. URL http://auai.org/uai2019/proceedings/papers/267.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Adrià Ruiz and Jakob Verbeek. Adaptative inference cost with convolutional neural mixture models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1872–1881, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

Frederick Tung, Srikanth Muralidharan, and Greg Mori. Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization. *arXiv preprint arXiv:1707.09102*, 2017.

Yunhe Wang, Chang Xu, Jiayan Qiu, Chao Xu, and Dacheng Tao. Towards evolutionary compression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2476–2485, 2018.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Learning sparsity and quantization jointly and automatically for neural network compression via constrained optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=HJ94fqApW`.

Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 8, 2019a.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019b.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1gMCsAqY7`.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. *arXiv preprint arXiv:2003.11142*, 2020.

Jihun Yun, Peng Zheng, Eunho Yang, Aurelie Lozano, and Aleksandr Aravkin. Trimming the ℓ_1 regularizer: Statistical analysis, optimization, and applications to deep learning. In *International Conference on Machine Learning*, pages 7242–7251, 2019.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

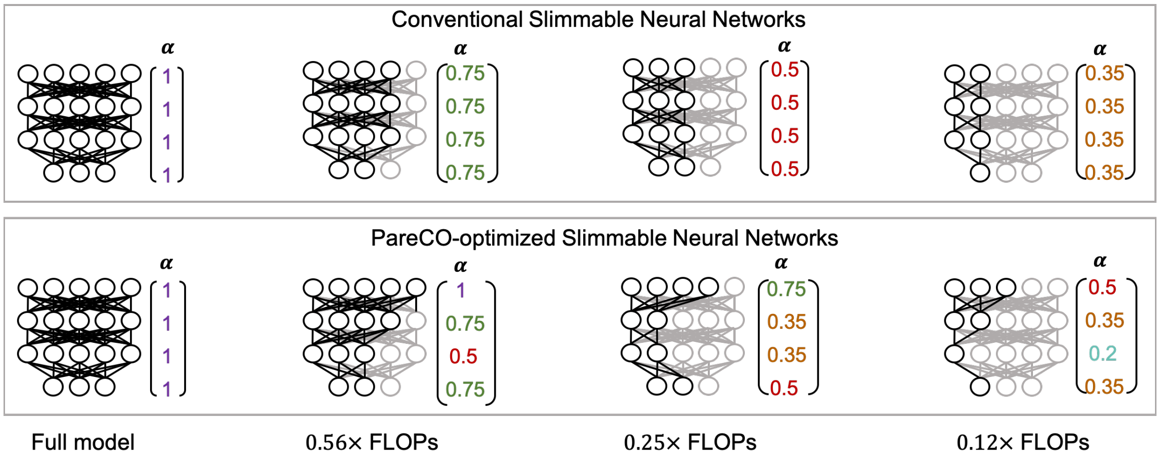


Figure 2: Conventional slimmable neural networks use a single width-multiplier for all the layers. We propose to optimize the widths together with the shared weights, which results in heterogeneous width-multipliers across different layers. $\boldsymbol{\alpha}$ is the vector of the width-multipliers.

Appendix A. Schematic of PareCO

A schematic view of the difference between the proposed and the conventional slimmable networks is shown in Figure 2.

Appendix B. Algorithm for PareCO

One can recover slimmable training (Yu and Huang [2019b]) by replacing lines 8 with randomly sampling a single width-multiplier for all the layers and setting $n = 1$ in line 13.

Algorithm 1: PareCO

```
   Input  : Model parameters $\boldsymbol{\theta}$, lower bound for width-multipliers $w_0 \in [0,1]$, number of full iterations
            $F$, number of gradient descent updates $n$, number of $\boldsymbol{\lambda}$ samples $M$
   Output : Trained parameter $\boldsymbol{\theta}$, approximate Pareto front $\mathcal{N}$
1  $\mathcal{H} = \{\}$        (Historical minimizers $\widehat{\boldsymbol{\alpha}}$)
2  for $i = 1...F$ do
3      $\boldsymbol{x}, y = $ sample_data()
4      $\boldsymbol{f}_{\text{CE}}, \boldsymbol{f}_{\text{FLOPs}} = f_{\text{CE}}(\mathcal{H}; \boldsymbol{\theta}, \boldsymbol{x}, y), f_{\text{FLOPs}}(\mathcal{H})$   (Calculate the objectives for each $\widehat{\boldsymbol{\alpha}} \in \mathcal{H}$)
5      $\boldsymbol{g} = $ BuildGP-UCB( $\mathcal{H}, \boldsymbol{f}_{\text{CE}}, \boldsymbol{f}_{\text{FLOPs}}$ )      (Build acquisition functions via BoTorch (Balandat
         et al. [2019]))
6      widths = []
7      for $m = 1...M$ do
8          $\widehat{\boldsymbol{\alpha}}, \mathcal{N} = $ PAS( $\boldsymbol{g}, \mathcal{H}, \boldsymbol{f}_{\text{CE}}, \boldsymbol{f}_{\text{FLOPs}}$ )     (Algorithm 2 in Appendix ??)
9          widths.append($\widehat{\boldsymbol{\alpha}}$)
10     end
11     $\mathcal{H} = \mathcal{H} \cup$ widths            (update historical data)
12     widths.append($\boldsymbol{w_0}$)       (smallest width for the sandwich rule in (Yu and Huang [2019b]))
13     for $j = 1...n$ do
14         SlimmableTraining( $\boldsymbol{\theta}$, widths )    (line 3-16 of Algorithm 1 in (Yu and Huang [2019b]) with
             provided widths)
15     end
16 end
```

Algorithm 2: Pareto-aware sampling (PAS)

```
Input  : Acquisition functions $\boldsymbol{g}$, historical data $\mathcal{H}$, $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}}$, search precision $\epsilon$
Output : channel configurations $\widehat{\boldsymbol{\alpha}}$, an approximate Pareto front $\mathcal{N}$
$\beta = 10^{-6}$    (A small positive number according to (Nakayama et al. [2009]))
$\boldsymbol{A}, \mathcal{N}$ = computeArea( $\mathcal{H}$, $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}}$ )    ( equation (3) )
$\tilde{f}_{\text{FLOPs}}$ = multinomial($\boldsymbol{A}$)    (Sample a target FLOPs)
$\lambda_{\text{FLOPs}}, \lambda_{\min}, \lambda_{\max} = 0.5, 0, 1$
while $|\frac{f_{FLOPs}(\widehat{\boldsymbol{\alpha}}) - \tilde{f}_{FLOPs}}{FullModelFLOPs}| > \epsilon$ do    // binary search
    $\widehat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \Big[\max_{i \in \{\text{CE,FLOPs}\}} \lambda_i (g_i(\boldsymbol{\alpha}) - \bar{g}_i) + \beta \sum_{i \in \{\text{CE,FLOPs}\}} \lambda_i g_i(\boldsymbol{\alpha})\Big]$
    if $f_{FLOPs}(\widehat{\boldsymbol{\alpha}}) > \tilde{f}_{FLOPs}$ then
        $\lambda_{\min} = \lambda_{\text{FLOPs}}$
        $\lambda_{\text{FLOPs}} = (\lambda_{\text{FLOPs}} + \lambda_{\max})/2$
    else
        $\lambda_{\max} = \lambda_{\text{FLOPs}}$
        $\lambda_{\text{FLOPs}} = (\lambda_1 + \lambda_{\min})/2$
    end
end
```

Appendix C. Related work

C.1 Slimmable neural networks

Slimmable neural networks (Yu et al. [2019]) enable multiple sub-networks with different compression ratios to be generated from a single network with one set of weights. This allows the FLOPs of network to be dynamically configurable at run-time without increasing the storage cost of the model weights. Based on this concept, better training methodologies have been proposed to enhance the performance of slimmable networks (Yu and Huang [2019b]). One can view a slimmable network as a dynamic computation graph where the graph can be constructed dynamically with different accuracy and FLOPs profiles. With this perspective, one can go beyond changing just the width of the network. For example, one can alter the network's sub-graphs (Ruiz and Verbeek [2019]), network's depth (Bolukbasi et al. [2017]; Elbayad et al. [2019]; Huang et al. [2017]; Li et al. [2019a]; Kaya et al. [2019]), and network's kernel sizes and input resolutions Cai et al. [2020]; Yu et al. [2020]. Complementing prior work primarily focusing on generalizing slimmable networks to additional architectural paradigms, our work provides the first principled multi-objective formulation for optimizing slimmable networks with tunable architecture decisions. While our analysis focuses on the network widths, our proposed formulation can be easily extended to other architectural parameters; we leave such instantiations to future work.

Relationship to slimmable training Conventional slimmable training (Yu and Huang [2019b]) is a special case of our proposed framework where $\boldsymbol{\lambda}$ is ignored. In this case, $\widehat{\boldsymbol{\alpha}}$ has the same value which is shared across all layers and which is obtained by sampling a width-multiplier from a uni-variate uniform distribution. Crucially, sampling $\widehat{\boldsymbol{\alpha}}$ this way does not optimize the trade-off front explicitly.

C.2 Neural architecture search

A slimmable neural network can be viewed as an instantiation of weight-sharing. In the literature for neural architecture search (NAS), weight-sharing is commonly adopted to reduce the search cost (Liu et al. [2018]; Stamoulis et al. [2019]; Guo et al. [2019]; Bender et al. [2018]; Berman et al. [2020]; Yu and Huang [2019a]). Specifically, NAS methods use weight-sharing as a proxy for evaluating the performance of the sub-networks to reduce the cost of iterative training and evaluation. However, NAS methods are concerned with the architecture of the network and the found network is re-trained from scratch, which is different from the weight-sharing mechanism adopted in slimmable networks where the weights are used for multiple networks during test time.

C.3 Channel pruning

Reducing the channel or filter counts for a pre-trained model is also known as channel pruning. In channel pruning, the goal is to maximize the accuracy of the pruned network subject to some resource constraints. Several studies have investigated how to better characterize redundant channels to prune them away. Channel pruning based on the magnitude of filter weights (Li et al. [2016]; He et al. [2019]), the magnitude of γ in batch normalization layer (Liu et al. [2017]; Ye et al. [2018]), and Taylor expansion (Molchanov et al. [2016, 2019]; Aflalo et al. [2020]) to the loss function have been investigated. Besides a post-processing perspective to channel pruning, prior work has also investigated channel pruning via an optimization lens. Specifically, channel pruning methods based on Lasso (Wen et al. [2016]; Liu et al. [2017]; Gordon et al. [2018]), trimmed Lasso (Yun et al. [2019]), stochastic ℓ_0 (Louizos et al. [2017b]), Bayesian compression (Louizos et al. [2017a]), soft filter pruning (He et al. [2018a]), and ADMM (Li et al. [2019b]; Yang et al. [2020]) have been developed. Liu et al. [2019b] later show that channel counts for different layers are more important for the performance of channel pruning. As a result, several studies have investigated pruning via an architecture search perspective. For example, using greedy algorithm (Yang et al. [2018]; Yu and Huang [2019a]), reinforcement learning (He et al. [2018b]), Bayesian optimization (Tung et al. [2017]; Ma et al. [2019]), dynamic programming (Berman et al. [2020]), and evolutionary algorithms (Chin et al. [2020]; Liu et al. [2019a]; Wang et al. [2018]) to search for the channel counts for each layer.

As shown across various channel pruning papers that a single pruning ratio for all the layers can be sub-optimal (Chin et al. [2020]; He et al. [2018b]; Liu et al. [2019a]; Gordon et al. [2018]; Liu et al. [2019b]; Yang et al. [2018]), it is natural to wonder: *can slimmable networks also benefit from non-uniform width-multipliers?* This question motivated us to develop a principled way to optimize network widths for slimmable neural networks. Crucially, compared to the channel pruning literature, our target is a slimmable neural network where its weights are shared across different sub-networks. This entails a different problem formulation. Specifically, in channel pruning, the weights of the network are optimized solely to improve the performance of the pruned network. In contrast, in a slimmable neural network, the weights of the network are optimized to improve the performance of multiple sub-networks.

Appendix D. Width parameterization

For ResNets with CIFAR, $\boldsymbol{\alpha}$ has six dimensions and is denoted by $\boldsymbol{\alpha}_{1:6} \in [0.316, 1]$, *i.e.*, one parameter for each stage and one for each residual connected layers in three stages. More specifically, the network is divided into three stages according to the output resolution, and as a result, there are three stages for all the ResNets designed for CIFAR. For example, in ResNet20, there are 7, 6, and 6 layers for each of the stages, respectively. Also, the layers that are added together via residual connection have to share the same width-multiplier, which results in one width-multiplier per stage for the layers that are connected via residual connections.

For MobileNetV2, $\boldsymbol{\alpha}_{1:25} \in [0.42, 1]$, and therefore there is one dimension for each independent convolutional layer. Note that while there are in total 52 convolutional layers in MobileNetV2, not all of them can be altered independently. More specifically, for layers that are added together via residual connection, their widths should be identical. Similarly, the depth-wise convolutional layer should have the same width as its preceding point-wise convolutional layers. The same logic applies to MobileNetV3, which has 47 convolutional layers (excluding squeeze-and-excitation layers) and $\boldsymbol{\alpha}_{1:22} \in [0.42, 1]$. In MobileNetV3, there are squeeze-and-excitation (SE) layers and we do not alter the width for the expansion layer in the SE layer. The output width of the SE layer is set to be the same as that of the convolutional layer where the SE layer is applied to. Crucially, there is no concept of expansion ratio for the inverted residual block in MobileNets in our width optimization. More specifically, the convolutional layer that acts upon expansion ratio is in itself just a convolutional layer with tunable width. Also, we do not quantize the width to be multiples of 8 as adopted in the previous work (Sandler et al. [2018]; Yu and Huang [2019b]). Due to these reasons, our 0.42× MobileNetV2 has 59 MFLOPs, which has the same FLOPs as the 0.35× MobileNetV2 in (Yu and Huang [2019b]; Sandler et al. [2018]).

Appendix E. Training hyperparameters

We use PyTorch Paszke et al. [2019] as our deep learning framework and we use BoTorch (Balandat et al. [2019]) for the implementation of MOBO-RS, which works seamlessly with PyTorch. More specifically, for the covariance function of Gaussian Processes, we use the commonly adopted Matérn Kernel Matérn [2013] without changing the default hyperparameters provided in BoTorch. Similarly, we use the default hyperparameter provided in BoTorch for the Upper Confidence Bound acquisition function. To perform the optimization of line 6 in Algorithm 2, we make use of the API "*optimize_ acqf*" provided in BoTorch.

For all the PareCO experiments, we set n such that PareCO only visits 1000 width configurations throughout the entire training ($|\mathcal{H}| = 1000$). Also, we set M to be 2, which follows the conventional slimmable training method (Yu and Huang [2019b]) that samples two width configurations in between the largest and the smallest widths.

CIFAR The training hyperparameters for the non-shared models are 0.1 initial learning rate, 200 training epochs, 0.0005 weight decay, 128 batch size, SGD with nesterov momentum, and cosine learning rate decay. The accuracy on the validation set is reported using the model at the final epoch. For slimmable training, we keep the same exact hyperparameters but train 2× longer compared to non-shared models, *i.e.*, 400 epochs.

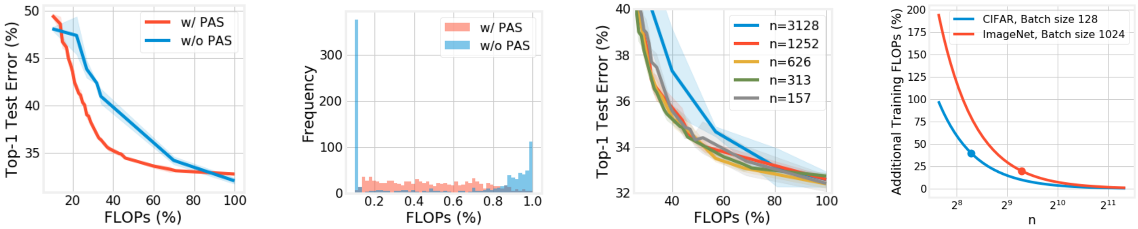
ImageNet Our training hyperparameters follow that of (Yu and Huang [2019b]). Specifically, we use initial learning rate of 0.5 with 5 epochs linear warmup (from 0 to 0.5), linear learning rate decay (from 0.5 to 0), 250 epochs, $4e^{-5}$ weight decay, 0.1 label smoothing, and we use SGD with 0.9 nesterov momentum. We use a batch size of 1024. For data augmentation, we use the "RandomResizedCrop" and "RandomHorizontalFlip" APIs in PyTorch. For MobileNetV2 we follow (Yu and Huang [2019b]) and use random scale between 0.25 to 1. For MobileNetV3, we use the default scale parameters, *i.e.*, from 0.08 to 1. The input resolution we use is 224. Besides scaling and horizontal flip, we follow (Yu and Huang [2019b]) and use color and lighting jitters data augmentataion with parameter of 0.4 for brightness, contrast, and saturation; and 0.1 for lighting. These augmentations can be found in the official repository of (Yu and Huang [2019b])[1]. The entire training is done using 8 NVIDIA V100 GPUs.

Appendix F. Ablation studies

In this section, we ablate the hyperparameters that are specific to PareCO to understand their impact. We use ResNet20 and CIFAR-100 for the ablation with the results summarized in Figure 3.

Pareto-aware sampling Without Pareto-aware sampling, one can also consider sampling $\boldsymbol{\lambda}$ uniformly from the Δ^{K-1}, which does not require any binary search and is easy to implement. However, the key issue with this sampling strategy is that uniform sampling $\boldsymbol{\lambda}$ does not necessarily imply uniform sampling in the objective space, *e.g.*, FLOPs. As shown in Figure 3a and Figure 3b, sampling in the objective space is more effective than sampling the $\boldsymbol{\lambda}$ space.

[1] `https://github.com/JiahuiYu/slimmable_networks/blob/master/train.py#L43`



(a) Impact of Pareto-aware sampling (PAS).

(b) Histogram of FLOPs for $\mathcal{H}$ w/ and w/o PAS.

(c) Performance for different n.

(d) Additional overhead over Slim for different n.

Figure 3: Ablation study for Pareto-aware sampling and the number of gradient descent updates per full iteration using ResNet20 and CIFAR-100. Experiments are conducted three times and we plot the mean and standard deviation.

Number of full iterations We reduce the number of full iterations by increasing the number of gradient descent updates. In previous experiments, we have $n = 313$, which results in $|\mathcal{H}| = 1000$. Here, we ablate n to $156, 626, 1252, 3128$ such that $|\mathcal{H}| = 2000, 500, 250, 100$, respectively. With larger n, the algorithm introduce a worse approximation since there are overall less iterations put into MOBO-RS. As shown in Figure 3c, we observe worse results with higher n. On the other hand, the improvement introduced by lower n saturates quickly. The training overhead of PareCO as a function of n compared to Slim is shown in Figure 3d where the dots are the employed n.