PareCO: Pareto-aware Channel Optimization for Slimmable Neural Networks

Ting-Wu Chin
tingwuc@cmu.edu
Carnegie Mellon University

Ari S. Morcos
arimorcos@fb.com
Facebook AI Research

Diana Marculescu
dianam@utexas.edu
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. 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.

ACM Reference Format:
Ting-Wu Chin, Ari S. Morcos, and Diana Marculescu. . PareCO: Pareto-aware Channel Optimization for Slimmable Neural Networks. In *DLP-KDD 2020: Workshop on Deep Learning Practice for High-Dimensional Sparse Data, August 24, 2020, San Diego, CA.* ACM, New York, NY, USA, 9 pages.

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* [48]. 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" [15]. 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 [46, 48], such a specification neglects that different layers affect the network's output differently [50] and have different FLOP requirements [9], 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 1.

The contributions of this work are three-fold. First, through a multi-objective optimization lens, we provide the first principled formulation for jointly optimizing the weights and widths of slimmable neural networks. The proposed formulation is general and can be applied to objectives other than prediction error and FLOPs [46, 48]. Second, we propose Pareto-aware Channel Optimization or PareCO, a novel algorithm which approaches the intractable problem formulation in an approximate fashion using stochastic gradient descent, of which the conventional slimmable neural networks is a special case. Quantitatively, PareCO improves conventional slimmable training by up to 2% and 1.9% in top-1 accuracy on the ImageNet dataset for MobileNetV2 and MobileNetV3, respectively. Finally, we perform extensive empirical analysis using 14 network and dataset combinations and find that less over-parameterized networks benefit more from jointly optimizing widths and weights.

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 $\boldsymbol{f(x)} = (f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x}))$ 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 $\boldsymbol{f(x)} < \boldsymbol{f(y)}$, for any $\boldsymbol{x} \in \mathcal{P}$ and $\boldsymbol{y} \notin \mathcal{P}$.*

DLP-KDD 2020, August 24, 2020, San Diego, California, USA
.

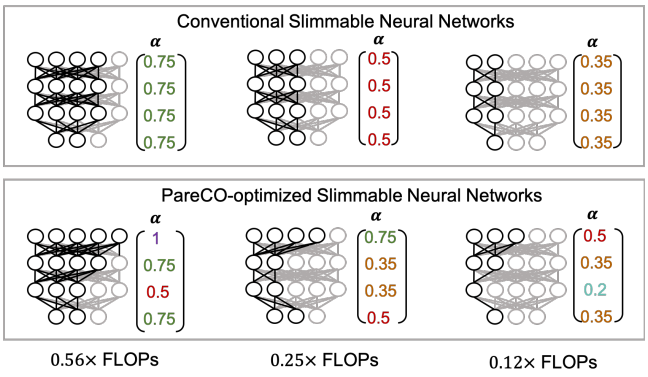


Figure 1: Conventional slimmable neural networks use a single width-multiplier for all the layers. We propose to optimize the widths together with the shared weights, which results in heterogeneous width-multipliers across different layers. $\boldsymbol{\alpha}$ is the vector of the width-multipliers.

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 2.1 (Augmented Tchebyshev Scalarization). *Define a scalarization of K objectives as*

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

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. (Section 1.3.3 in [32])

With Lemma 2.1, 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 $\lambda_{\text{CE}} \rightarrow 1$ and $\lambda_{\text{FLOPs}} \rightarrow 0$, 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. Now, we can formally define our problem of interest:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}} \mathbb{E}_{\boldsymbol{\lambda}\sim\mathcal{L}} [\min_{\boldsymbol{\alpha}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{\alpha}; \boldsymbol{\theta}, \boldsymbol{x}, y)], \tag{2}$$

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 inner minimization can be intractable since the objective 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) [34] to approximate the inner optimization problem.

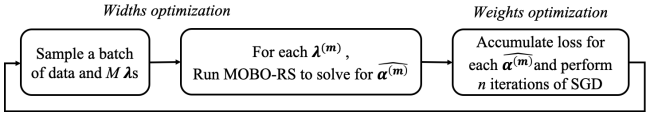


Figure 2: The PareCO framework for training slimmable networks.

2.2 PareCO: Pareto-aware channel optimization

The proposed framework, as shown in Figure 2, 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 avoid superscripts when possible in the sequel.

Relationship to slimmable training. Conventional slimmable training [46] is a special case of this 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 univariate uniform distribution. Crucially, sampling $\widehat{\boldsymbol{\alpha}}$ this way does not optimize the trade-off front explicitly. Next, we describe the proposed $\boldsymbol{\lambda}$ sampling process, the implementation for MOBO-RS, and the weight optimization process.

Pareto-aware sampling. Each $\boldsymbol{\lambda}$ determines a point-of-interest on the Pareto curve when the corresponding augmented Tchebyshev scalarized objective is solved. Thus, the distribution of $\boldsymbol{\lambda}$ determines the distribution of the objectives $f_i(\widehat{\boldsymbol{\alpha}})$ and the *mapping between the two is non-trivial.*

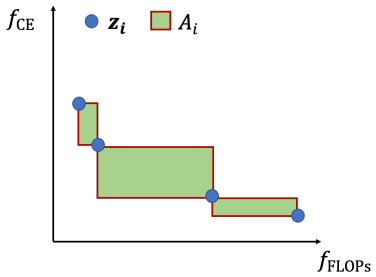


Figure 3: An example of the various notations used in Pareto-aware sampling where z_i is a minimizer in the approximate Pareto frontier and A_i quantifies under-exploration for the regions between z_i and z_{i+1}.

To sample diverse solutions on the Pareto curve, we make use of the width configurations obtained so far across full iterations $\mathcal{H}$ (a full iteration is one iteration of Figure 2) 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,

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

where $\boldsymbol{z} \in \mathcal{N}$ is ordered solutions sorted in ascending order according to $f_{\text{FLOPs}}(\cdot)$. We visualize an example in Figure 3.

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 computationally costly. To reduce computational 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 $\mathcal{H} = \{\widehat{\boldsymbol{\alpha_1}}, \ldots, \widehat{\boldsymbol{\alpha_{t-1}}}\}$. 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 computationally 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 D.

PareCO. Based on this preamble, we propose PareCO, which is detailed in Algorithm 1. It follows Figure 2 by using Pareto-aware sampling, one-step MOBO-RS, and a large n. 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 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. One can recover slimmable training [46] 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 [2]*)
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*)
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 [46]*)
13 **for** *j = 1...n* **do**
14 SlimmableTraining($\boldsymbol{\theta}$, widths) (*line 3-16 of Algorithm 1 in [46]* ***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 ϵ
Output : channel configurations $\widehat{\boldsymbol{\alpha}}$, an approximate Pareto front $\mathcal{N}$

1 $\beta = 10^{-6}$ (*A small positive number according to [32]*)
2 $\boldsymbol{A}, \mathcal{N}$ = computeArea($\mathcal{H}, \boldsymbol{f}_{\text{CE}}, \boldsymbol{f}_{\text{FLOPs}}$) (*equation (3)*)
3 $\tilde{f}_{\text{FLOPs}}$ = multinomial($\boldsymbol{A}$) (*Sample a target FLOPs*)
4 $\lambda_{\text{FLOPs}}, \lambda_{\min}, \lambda_{\max}$ = 0.5, 0, 1
5 **while** $|\frac{f_{FLOPs}(\widehat{\boldsymbol{\alpha}}) - \tilde{f}_{FLOPs}}{FullModelFLOPs}| > \epsilon$ **do** // binary search
6 $\widehat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \left[\max_{i \in \{\text{CE,FLOPs}\}} \lambda_i (g_i(\boldsymbol{\alpha}) - \bar{g}_i) + \beta \sum_{i \in \{\text{CE,FLOPs}\}} \lambda_i g_i(\boldsymbol{\alpha})\right]$
7 **if** $f_{FLOPs}(\widehat{\boldsymbol{\alpha}}) > \tilde{f}_{FLOPs}$ **then**
8 $\lambda_{\min} = \lambda_{\text{FLOPs}}$
9 $\lambda_{\text{FLOPs}} = (\lambda_{\text{FLOPs}} + \lambda_{\max})/2$
10 **else**
11 $\lambda_{\max} = \lambda_{\text{FLOPs}}$
12 $\lambda_{\text{FLOPs}} = (\lambda_1 + \lambda_{\min})/2$
13 **end**
14 **end**

3 EXPERIMENTS

For all the PareCO experiments in this section, 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 [46] that samples two width configurations in between the largest and the smallest widths. Other information details are listed in Appendix.

First, we are interested in the following question: *Do PareCO-optimized models (PareCO) always outperform conventional slimmable networks (Slim) [46]?* 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 4, PareCO improves Slim in most cases for CIFAR-100 while PareCO performs similarly

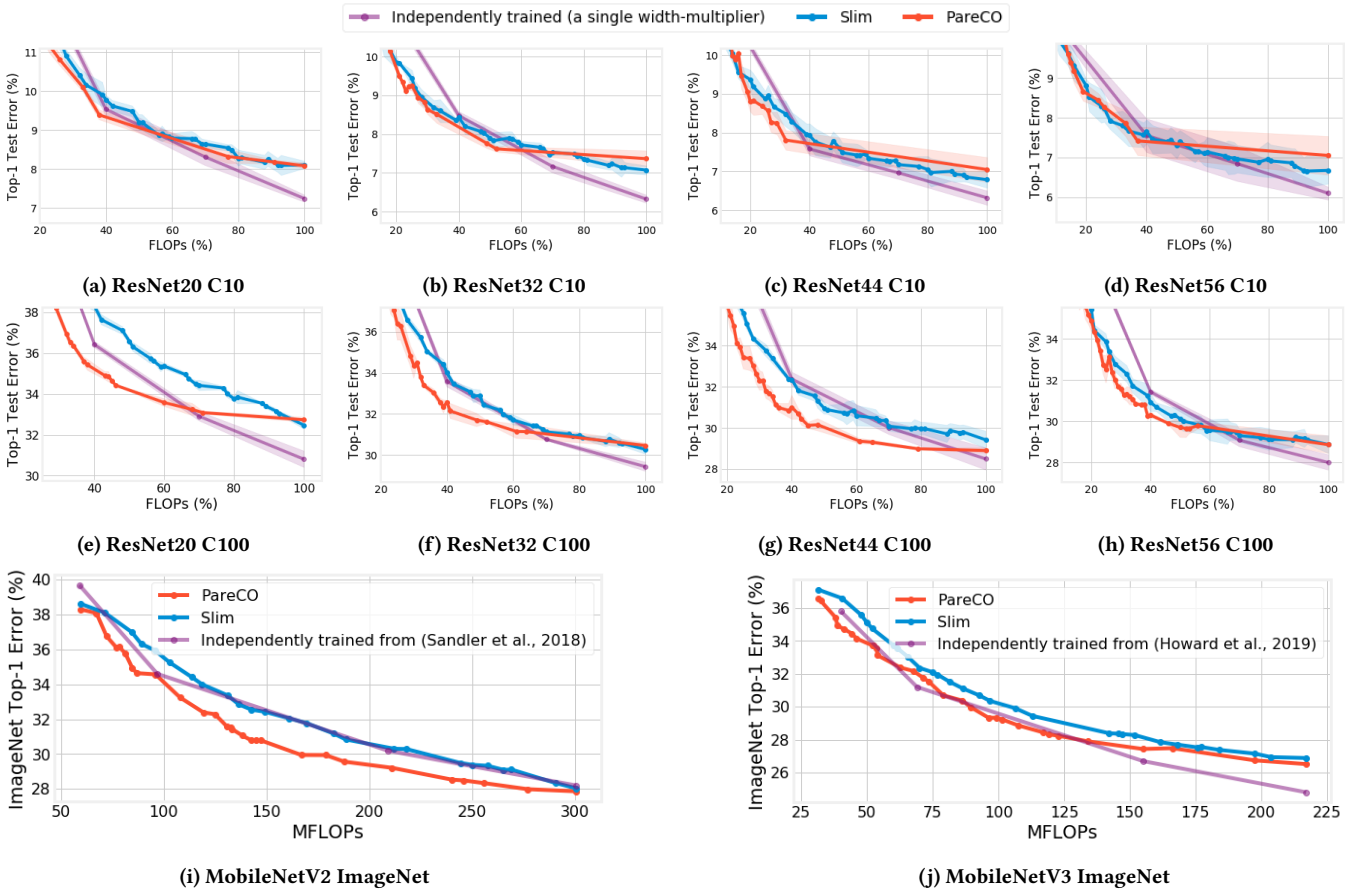


Figure 4: Comparisons among PareCO and Slim. C10 and C100 denote CIFAR-10/100. For the CIFAR dataset, we perform three trials for each method and plot the mean and standard deviation. PareCO is better or comparable to Slim.

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 4d. 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 4e. 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 [37] and MobileNetV3 [14]. As shown in Figure 4i and Figure 4j, we observe the similar trend that PareCO outperforms Slim. Quantitatively, up to 2% and 1.9% top-1 accuracy improvements can be achieved for MobileNetV2 and MobileNetV3, respectively. Compared to independently trained models (no weight-sharing) using a single width-multiplier, PareCO manages to perform better in low-FLOPs regimes with optimized widths.

Numerical results are detailed in Appendix E. Thanks to the large n and inner optimization approximation, PareCO only incurs approximately 20% additional overhead compared to Slim in training.

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. In our empirical analysis, we extensively verify the effectiveness of PareCO over conventional slimmable training on 14 dataset and network combinations. Moreover, we find that less over-parameterized networks benefit more from joint channel and weight optimization. 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

[1] Yonathan Aflalo, Asaf Noy, Ming Lin, Itamar Friedman, and Lihi Zelnik. 2020. Knapsack Pruning with Inner Distillation. *arXiv preprint arXiv:2002.08258* (2020).

[2] Maximilian Balandat, Brian Karrer, Daniel R Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2019. BoTorch: Programmable Bayesian Optimization in PyTorch. *arXiv preprint arXiv:1910.06403* (2019).

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and Simplifying One-Shot Architecture Search. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 550–559. http://proceedings.mlr.press/v80/bender18a.html

[4] Maxim Berman, Leonid Pishchulin, Ning Xu, Gérard Medioni, et al. 2020. AOWS: Adaptive and optimal network width search with latency constraints. *Proceedings IEEE CVPR* (2020).

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

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HylxE1HKwS

[7] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. 2020. Towards Efficient Model Compression via Learned Global Ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[8] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive Transformer. *arXiv preprint arXiv:1910.10073* (2019).

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

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

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

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

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

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

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

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

[17] Yiğitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*. Long Beach, CA.

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

[19] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. 2019. Improved Techniques for Training Adaptive Deep Networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 1891–1900.

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

[21] Zhengang Li, Yifan Gong, Xiaolong Ma, Sijia Liu, Mengshu Sun, Zheng Zhan, Zhenglun Kong, Geng Yuan, and Yanzhi Wang. 2020. SS-Auto: A Single-Shot, Automatic Structured Weight Pruning Framework of DNNs with Ultra-High Efficiency. *arXiv preprint arXiv:2001.08839* (2020).

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

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

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

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2019. Rethinking the Value of Network Pruning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJlnB3C5Ym

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

[27] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint arXiv:1712.01312* (2017).

[28] Xingchen Ma, Amal Rannen Triki, Maxim Berman, Christos Sagonas, Jacques Cali, and Matthew B Blaschko. 2019. A Bayesian Optimization Framework for Neural Network Compression. In *Proceedings of the IEEE International Conference on Computer Vision*. 10274–10283.

[29] Bertil Matérn. 2013. *Spatial variation*. Vol. 36. Springer Science & Business Media.

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

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

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

[33] Wei Niu, Pu Zhao, Zheng Zhan, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Towards Real-Time DNN Inference on Mobile Platforms with Model Pruning and Compiler Optimization. *arXiv preprint arXiv:2004.11250* (2020).

[34] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. 2019. A Flexible Framework for Multi-Objective Bayesian Optimization using Random Scalarizations. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 267. http://auai.org/uai2019/proceedings/papers/267.pdf

[35] 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. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8026–8037. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[36] Adrià Ruiz and Jakob Verbeek. 2019. Adaptative Inference Cost With Convolutional Neural Mixture Models. In *Proceedings of the IEEE International Conference on Computer Vision*. 1872–1881.

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

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

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

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

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

[42] Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. 2020. 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*.

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

[44] Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. 2018. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJ94fqApW

[45] Jiahui Yu and Thomas Huang. 2019. AutoSlim: Towards One-Shot Architecture Search for Channel Numbers. *arXiv preprint arXiv:1903.11728* 8 (2019).

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

[47] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models. *arXiv preprint arXiv:2003.11142* (2020).

[48] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1gMCsAqY7

[49] Jihun Yun, Peng Zheng, Eunho Yang, Aurelie Lozano, and Aleksandr Aravkin. 2019. Trimming the ℓ_1 Regularizer: Statistical Analysis, Optimization, and Applications to Deep Learning. In *International Conference on Machine Learning*. 7242–7251.

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

A RELATED WORK

A.1 Slimmable neural networks

Slimmable neural networks [48] 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 [46]. 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 [36], network's depth [5, 8, 16, 17, 19], and network's kernel sizes and input resolutions [6, 47]. 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.

A.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 [3, 4, 10, 22, 38, 45]. Specifically, NAS methods use weight-sharing as a proxy for evaluating the performance of the sub-networks to reduce the computational 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.

A.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 [13, 18], the magnitude of γ in batch normalization layer [23, 44], and Taylor expansion [1, 30, 31] 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 [9, 23, 41], trimmed Lasso [49], stochastic ℓ_0 [27], Bayesian compression [26], soft filter pruning [11], and ADMM [20, 21, 33, 42] have been developed. Liu *et al.* [25] 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 [43, 45], reinforcement learning [12], Bayesian optimization [28, 39], dynamic programming [4], and evolutionary algorithms [7, 24, 40] 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 [7, 9, 12, 24, 25, 43], 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.

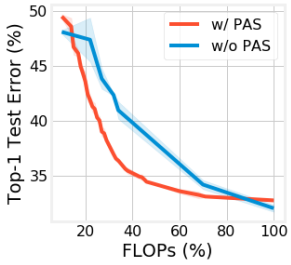
B WIDTH PARAMETERIZATION

For ResNets with CIFAR, $\boldsymbol{\alpha} \in [0.316, 1]^6$. 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.

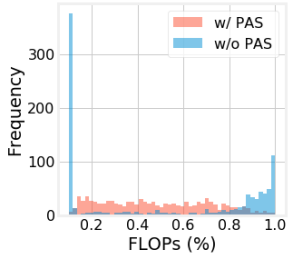
For MobileNetV2, $\boldsymbol{\alpha} \in [0.35, 1]^{25}$, which represents one dimension for each independent convolutional layer. Note that while there are in total 52 convolutional layers in MobileNetV2, not all of them can be altered independently. More specifically, for layers that are added together via residual connection, their width 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 $\boldsymbol{\alpha} \in [0.35, 1]^{22}$. 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 itself just a convolutional layer with tunable width. Also, do not quantize the width to be multiples of 8 as adopted in the previous work [37, 46].

C TRAINING HYPERPARAMETERS

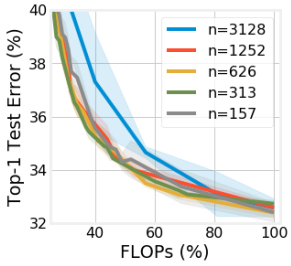
We use PyTorch [35] as our deep learning framework and we use BoTorch [2] for the implementation of MOBO-RS, which works seamlessly with PyTorch. More specifically, for the covariance function of Gaussian Processes, we use the commonly adopted Matérn Kernel [29] without changing the default hyperparameters 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.



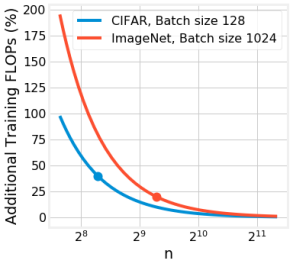
(a) Impact of Pareto-aware sampling (PAS).



(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 5: 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.

CIFAR. The training hyperparameters for the independent 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 exact same hyperparameters but train 2× longer compared to independent models, *i.e.*, 400 epochs.

ImageNet. For MobileNetV2, the training hyperparameters are a learning rate of $0.125 \times \frac{\text{batch size}}{256}$ with 5 epochs linear warmup, linear learning rate decay, 250 epochs, $4e^{-5}$ weight decay, 0.1 label smoothing, a batch size of 1024, and we use SGD with 0.9 momentum. The data augmentation strategy follows [46]. For MobileNetV3, we use a learning rate of $0.05 \times \frac{\text{batch size}}{256}$ with 5 epochs linear warmup, cosine learning rate decay, 250 epochs, $4e^{-5}$ weight decay, 0.1 label smoothing, a batch size of 1536, and we use SGD with 0.9 momentum. The data augmentation only includes random crop and random horizontal flips. The image resolution is fixed at 224. We note that weight averaging (exponential moving average) is adopted in the original MobileNetV3 training pipeline [14] and we have not used it. The entire training is done using 8 NVIDIA V100 GPUs.

The hyperparameters for MobileNetV2 mainly follow [46] while the hyperparameters for MobileNetV3 mainly follow the hyperparameters used for training standalone MobileNetV2 in PyTorch[1] and we have not tuned these hyperparameters.

D 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 5.

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 5a and Figure 5b, sampling in the objective space is more effective than sampling the $\boldsymbol{\lambda}$ space.

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 5c, 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 5d where the dots are the employed n.

E NUMERICAL RESULTS FOR IMAGENET

We summarize the numbers from Figure 4i and Figure 4j in Table 1 for future work to compare easily.

[1] https://github.com/d-li14/mobilenetv2.pytorch

MobileNetV2				MobileNetV3			
MFLOPs	Independently trained [37]	Slim	PareCO	MFLOPs	Independently trained [14]	Slim	PareCO
59	60.3	61.4	**61.5** (+0.1)	40	**64.2**	-	-
71	-	61.9	**63.0** (+1.1)	42	-	65.8	**65.9** (+0.1)
84	-	63.0	**64.6** (+1.6)	51	-	66.3	**66.6** (+0.3)
95	-	64.0	**65.1** (+1.1)	60	-	67.2	**67.7** (+0.5)
97	65.4	-	-	69	**68.8**	-	-
102	-	64.7	**65.5** (+0.8)	73	-	68.1	**68.8** (+0.7)
136	-	67.1	**68.2** (+1.1)	84	-	69.0	**70.0** (+1.0)
149	-	67.6	**69.1** (+1.5)	118	-	71.0	**71.4** (+0.4)
169	-	68.2	**69.9** (+1.7)	121	-	71.0	**71.6** (+0.6)
209	69.8	-	-	155	**73.3**	-	-
212	-	69.7	**70.6** (+0.9)	168	-	72.7	**72.8** (+0.1)
244	-	70.5	**71.0** (+0.5)	183	-	73.0	**73.2** (+0.2)
300	71.8	72.0	**72.1** (+0.1)	217	**75.2**	73.5	73.7 (+0.2)

Table 1: MobileNetV2 and MobileNetV3 on ImageNet. The number in the parenthesis for PareCO are the improvements compared to the corresponding Slim. Bold represents the highest accuracy of a given FLOPs.