PARECO: PARETO-AWARE CHANNEL OPTIMIZATION FOR SLIMMABLE NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Slimmable neural networks 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. They have been proposed recently for resource-constrained settings such as mobile devices. 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. Hence, developing a principled approach for deciding width-multipliers across different layers could potentially improve the performance of slimmable networks. To allow for heterogeneous width-multipliers across different layers, 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. We perform extensive empirical analysis with 14 network and dataset combinations and find that less over-parameterized networks benefit more from a joint channel and weight optimization than extremely over-parameterized networks. Quantitatively, improvements up to 1.7% and 1% in top-1 accuracy on the ImageNet dataset can be attained for MobileNetV2 and MobileNetV3, respectively. Our results highlight the potential of optimizing the channel counts for different layers jointly with the weights for slimmable networks.

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). This is in stark contrast to channel pruning methods (Berman et al., 2020; Yu & Huang, 2019a; Guo et al., 2020; Molchanov et al., 2019) that aim for a small standalone model. Slimmable 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 ConvNet-powered 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 $0.35\times$ 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 & 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. In a similar setting, as demonstrated in the model pruning literature (Gordon et al., 2018; Liu et al., 2019b; Morcos et al., 2019; Renda et al., 2020), having different pruning ratios for different layers of

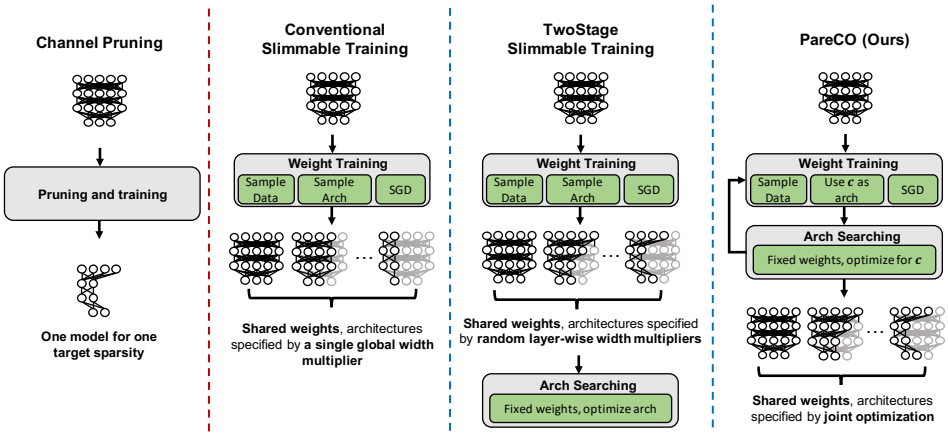


Figure 1: Schematic overview comparing our proposed method with existing alternatives and channel pruning. Channel pruning has a fundamentally different goal compared to ours, *i.e.*, training slimmable nets. PareCO jointly optimizes both the architectures and the shared weights.

the network can further improve results over a single ratio across layers. This raises an interesting question: *How should we obtain these non-uniform widths for slimmable nets?*

To achieve non-uniform width-multipliers across layers, one can consider using techniques from the neural architecture search (NAS) literature (Cai et al., 2020; Yu et al., 2020), which we call *TwoStage* training. Specifically, first train a supernet with weight-sharing by uniformly sampling width-multiplier for each layer. After this procedure converges, use multi-objective optimization methods to search for width given the trained weights. However, width optimization has a much larger design space than that considered in existing methods for NAS. Specifically, each layer can have hundreds of choices (since there are hundreds of channels for each layer). This makes it unclear if such a training technique is suitable for channel optimization[1].

As an alternative to existing techniques, 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 differences among the conventional slimmable training, TwoStage training, and our proposed method 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 (Yu & Huang, 2019b; Yu et al., 2019). 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 training method proposed for universally slimmable neural networks (Yu & Huang, 2019b) is a special case. Finally, we perform extensive empirical analysis using 14 network and dataset combinations and demonstrate the effectiveness of the proposed algorithm over existing techniques.

2 Related Work

2.1 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 & Huang, 2019b). One can view a slimmable network as a dynamic

[1]Both OFA (Cai et al., 2020) and BigNAS (Yu et al., 2020) mainly use the pre-defiend channel counts and search for kernel sizes, depth, and input resolution. Specifically, channel counts refer to expansion ratios only for OFA while BigNAS only considers a small range of channel counts near the pre-defined ones.

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 & 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.

2.2 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 & Huang, 2019a). 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.

2.3 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 find a single small model that maximizes the accuracy while satisfying some resource constraints. Several studies have investigated how to better characterize redundant channels in a post-training fashion given a pre-trained model (Li et al., 2016; He et al., 2019; Liu et al., 2017; Ye et al., 2018; Molchanov et al., 2016; 2019; Aflalo et al., 2020). 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; Yun et al., 2019), stochastic ℓ_0 (Louizos et al., 2017), and ADMM (Li et al., 2019b; Yang et al., 2020) have been developed. Liu *et al.* (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 (Yu & Huang, 2019a; Berman et al., 2020; Ma et al., 2019; Chin et al., 2020; Liu et al., 2019a).

While channel pruning also optimizes for non-uniform widths, the goal of channel pruning is crucially different from ours. The key difference is that channel pruning concerns about a single pruned model while slimmable neural networks require a set of models to be trained using weight sharing. As a result, it is not clear if those techniques are suitable for finding non-uniform widths for slimmable nets. Moreover, using channel pruning to obtain a set of pruned models can be computationally expensive. Specifically, the state-of-the-art differentiable pruning method takes roughly $2\times$ the training time for searching the channel counts for a given target FLOPs (Guo et al., 2020). That is, if we want 20 uniformly distributed FLOPs between 20% and 100%, it takes roughly $24\times$ the training time of the 100% FLOPs network just to obtain architectures. In contrast, our formulation is targeting slimmable nets directly with the training cost similar to that of conventional slimmable training, which is $4\times$ the training time of a standalone model.

3 Methodology

In this work, we are interested in jointly optimizing the network widths and network weights. Ultimately, when evaluating the performance of a slimmable neural network, we care about the trade-off curve between multiple objectives, *e.g.*, theoretical speedup and accuracy. This trade-off curve is formed by evaluating multiple networks with width configurations sampled from a width sampling distribution. Viewed from this perspective, the sampling distribution should be optimized in such a way that the resulting networks have a better trade-off curve (*i.e.*, larger area under curve), which is a multi-objective optimization problem. This section formalizes this idea and provides an algorithm to solve it in an approximate fashion.

3.1 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 $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x}))$ 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 $\boldsymbol{f}(\boldsymbol{x}) < \boldsymbol{f}(\boldsymbol{y})$, 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 3.1 (Augmented Tchebyshev Scalarization (Section 1.3.3 in (Nakayama et al., 2009))) *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.

The second term of equation (1) is the commonly used weighted sum scalarization and it can be depicted as a line in the objectives space. However, minimizing it alone is not sufficient if the Pareto curve is non-convex (Nakayama et al., 2009). This calls for the first term, which can be depicted as the axis-aligned lines of a non-positive orthant in the objectives space. With Lemma 3.1, 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 $\lambda_{\text{CE}} \to 1$ 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. In contrast, if taking $\lambda_{\text{CE}} \to 0$ and $\lambda_{\text{FLOPs}} \to 1$, we are then effectively seeking width configurations that minimize FLOPs. With the scalarization, we have the following theorem for summarizing the area under curve quantitatively:

Theorem 3.2 *Optimizing network weights $\boldsymbol{\theta}$ to minimize area under Pareto curve formed by cross-entropy loss (f_{CE}) and FLOPs (f_{FLOPs}) can be done approximately with the following objective:*

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{m=1}^{M} f_{CE}(\alpha(\boldsymbol{g}(\boldsymbol{z}^{(\boldsymbol{m})})), \boldsymbol{\theta}, \boldsymbol{x}, y), \tag{2}$$

where $\boldsymbol{x}, y$ are sampled training input and label, M is the number of Monte Carlo samples, $\boldsymbol{z}^{(\boldsymbol{m})}$ is a FLOPs target sampled uniformly between the lowest and the highest FLOPs, $\alpha(\cdot)$ maps from $\boldsymbol{\lambda}$ to the corresponding Pareto channel configuration (i.e., $\alpha(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{c}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{c}, \boldsymbol{\theta}, \boldsymbol{x}, y)$), and $g(\cdot)$ maps a certain FLOPs back to the corresponding $\boldsymbol{\lambda}$, (i.e., $g = (f_{FLOPs} \circ \alpha)^{-1}$).

The proof is in Appendix A. Intuitively speaking, the objective is to minimize the cross entropy loss for the Pareto-optimal channel configurations (given the current $\boldsymbol{\theta}, \boldsymbol{x}, y$) across different FLOPs uniformly. While equation (2) precisely defines our goal, solving $g(\cdot)$ and $\alpha(\cdot)$ can be intractable since the functions are usually highly non-convex with respect to channel configurations and do not have analytical gradient information that admits first-order optimization algorithms. To tackle these challenges, we propose to model both f_{CE} and f_{FLOPs} via Gaussian Processes (GPs) (Rasmussen, 2003). This allows $\alpha(\cdot)$ to be approximated via Bayesian Optimization (BO) (Srinivas et al., 2009) with low overhead, which in turn enables $g(\cdot)$ to be implemented using binary search with $f_{\text{FLOPs}} \circ \alpha$. Concretely, binary search is done by repeating the following: start with $\boldsymbol{\lambda}_t$, solve $\alpha(\boldsymbol{\lambda}_t)$ using BO, and decrease or increase $\boldsymbol{\lambda}_{t+1}$ depending on the FLOPs of the current architecture $f_{\text{FLOPs}}(\alpha(\boldsymbol{\lambda}))$.

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 $\mathcal{N}$

$\mathcal{H} = \{\}$ (*Historical minimizers* $\widehat{\boldsymbol{c}}$)
for *i = 1...F* **do**
  $\boldsymbol{x}, y$ = sample_data()
  $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}} = f_{\text{CE}}(\mathcal{H}; \boldsymbol{\theta}, \boldsymbol{x}, y)$, $f_{\text{FLOPs}}(\mathcal{H})$ (*Calculate the objectives for each* $\widehat{\boldsymbol{c}} \in \mathcal{H}$)
  $\boldsymbol{g}$ = BuildGP-UCB($\mathcal{H}$, $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}}$) (*Build acquisition func. via BoTorch (Balandat et al., 2019)*)
  widths = []
  for *m = 1...M* **do**
    $\widehat{\boldsymbol{c}}, \mathcal{N}$ = BOBS($\boldsymbol{g}, \mathcal{H}, \boldsymbol{f}_{\text{CE}}, \boldsymbol{f}_{\text{FLOPs}}$) (*Algorithm 2*)
    widths.append($\widehat{\boldsymbol{c}}$)
  end
  $\mathcal{H} = \mathcal{H} \cup$ widths (*update historical data*)
  widths.append($\boldsymbol{w_0}$) (*smallest width for the sandwich rule in (Yu & Huang, 2019b)*)
  for *j = 1...n* **do**
    SlimmableTraining($\boldsymbol{\theta}$, widths) (*line 3-16 of Algorithm 1 in (Yu & Huang, 2019b)*)
  end
  $\mathcal{N}$=nonDominatedSort($\mathcal{H}$, $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}}$)
end

3.2 Approximation via multi-objective Bayesian optimization

The goal is to approximate $\alpha(\cdot)$ and $g(\cdot)$ quickly with good quality. In this realm, Bayesian Optimization is known to be sample-efficient for doing so. Specifically, we are hoping to solve $\arg\min_{\boldsymbol{c}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{c}, \boldsymbol{\theta}, \boldsymbol{x}, y)$ efficiently using Bayesian Optimization where $\boldsymbol{c}$ here denotes the channel configurations. Instead of modeling $\mathcal{T}_{\boldsymbol{\lambda}}$ directly, we propose to model f_{CE} and f_{FLOPs} using two GPs. This design choice is mainly based on the fact that we would like to solve $g(\cdot)$ using binary search. More specifically, by having two GPs, the GPs can be reused across different $\boldsymbol{\lambda}$. On the other hand, if we directly model $\mathcal{T}_{\boldsymbol{\lambda}}$, we have to fit a GP for every $\boldsymbol{\lambda}$, which can be costly.

Having a separate GP for each objective of interest turns out to have no regret for optimization if a proper acquisition function is adopted (Paria et al., 2019). As a result, we follow Paria et al. (2019) and adopt the Upper Confidence Bound (UCB) (Srinivas et al., 2009) acquisition function for optimizing the problem of interest. To optimize the target objective, Bayesian Optimization proceeds sequentially. To begin, we start with training data $\mathcal{H}_{t-1} = \{\widetilde{\boldsymbol{c_1}}, \ldots, \widetilde{\boldsymbol{c_{t+o}}}\}$ and their function responses $f_i(\widetilde{\boldsymbol{c_\ell}})\ \forall\ i \in \{\text{CE}, \text{FLOPs}\}, \ell \in [t+o]$ where t denotes the current timestamp in the sequential optimization and o is the initial size of the training data. Then, GPs are fitted for both objectives and acquisition functions for both GPs are combined using the augmented Tchebyshev scalarization with a specified $\boldsymbol{\lambda}$, which is then optimized to obtain the next point $\boldsymbol{c_{t+o+1}}$. This minimization is tractable because it minimizes the surrogate function instead of the unknown function. Under properly set hyperparameters for UCB, it is known that this procedure introduces no regret (Paria et al., 2019). In other words, if we allocate enough time for BO, *i.e.*, $t \to \infty$, and set the output of BO to be $\widehat{\boldsymbol{c}} \stackrel{\text{def}}{=} \widetilde{\boldsymbol{c_{t+o}}}$, the method provides a close approximation, *i.e.*, $\mathcal{T}_{\boldsymbol{\lambda}}(\widehat{\boldsymbol{c}}, \boldsymbol{\theta}, \boldsymbol{x}, y) \approx \min_{\boldsymbol{c}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{c}, \boldsymbol{\theta}, \boldsymbol{x}, y)$.

3.3 Approximation via temporal similarity

Equipped with BO, solving equation (2) can be done by running the following three steps iteratively: (1) sample M target FLOPs, (2) solve the corresponding channel configurations using BO with binary search, and (3) perform a step of stochastic gradient descent. While BO provides a tractable mean for us to solve for Pareto-optimal channel configurations, it is still computationally expensive to run BO for every iteration of stochastic gradient descent. Thus, we propose to exploit temporal similarity for making the optimization practical. Specifically, we notice that $\boldsymbol{\theta}$ would not be drastically different across a few iterations since gradient descent itself relies on first-order Taylor approximation. As a result, we propose the following two approximations. First, instead of having many sequential queries in one subroutine of Bayesian optimization, we only perform one query and store the query

to $\mathcal{H}$ for future Bayesian optimization. Note that for each query in BO, the cross-entropy loss will be reevaluated for each $\boldsymbol{c} \in \mathcal{H}$ to build faithful Gaussian Processes. Our second approximation is to perform n SGD updates as opposed to one before another query for Bayesian optimization.

PareCO Based on this preamble, we present our algorithm, PareCO, in Algorithm 1. 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 target FLOPs and solve for the corresponding widths (*i.e.*, $\widehat{\boldsymbol{c}}$) via binary search with one query of BO, and (3) perform n gradient descent steps using the solved widths. One can recover slimmable training (Yu & Huang, 2019b) by replacing lines 8 with randomly sampling a single width-multiplier for all the layers and setting $n = 1$ in line 13. The outputs of PareCO are both the shared weights and the Pareto-optimal widths. To obtain the Pareto-optimal widths, we use non-dominated sort based on the trainnig loss and FLOPs for $c \in \mathcal{H}$.

Algorithm 2: Bayesian Optimization with Binary Search (BOBS)

Input : Acquisition functions $\boldsymbol{g}$, historical data $\mathcal{H}$, $\boldsymbol{f}_{\text{CE}}$, $\boldsymbol{f}_{\text{FLOPs}}$, search precision ϵ
Output : channel configurations $\widehat{\boldsymbol{c}}$

1 $\beta = 10^{-6}$ (*A small positive number according to (Nakayama et al., 2009)*)
2 $\tilde{f}_{\text{FLOPs}}$ = Uniform($f_{\text{FLOPs,min}}$, $f_{\text{FLOPs,max}}$) (*Sample a target FLOPs*)
3 λ_{FLOPs}, $\lambda_{\min}$, $\lambda_{\max}$ = 0.5, 0, 1
4 **while** $|\frac{f_{FLOPs}(\widehat{\boldsymbol{c}}) - \tilde{f}_{FLOPs}}{FullModelFLOPs}| > \epsilon$ **do** `// binary search`
5 $\quad \widehat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}} \left[\max_{i \in \{\text{CE,FLOPs}\}} \lambda_i (g_i(\boldsymbol{c}) - \bar{g}_i) + \beta \sum_{i \in \{\text{CE,FLOPs}\}} \lambda_i g_i(\boldsymbol{c})\right]$
6 $\quad$ **if** $f_{FLOPs}(\widehat{\boldsymbol{c}}) > \tilde{f}_{FLOPs}$ **then**
7 $\quad\quad \lambda_{\min} = \lambda_{\text{FLOPs}}$
8 $\quad\quad \lambda_{\text{FLOPs}} = (\lambda_{\text{FLOPs}} + \lambda_{\max})/2$
9 $\quad$ **else**
10 $\quad\quad \lambda_{\max} = \lambda_{\text{FLOPs}}$
11 $\quad\quad \lambda_{\text{FLOPs}} = (\lambda_{\text{FLOPs}} + \lambda_{\min})/2$
12 $\quad$ **end**
13 **end**

4 EXPERIMENTS

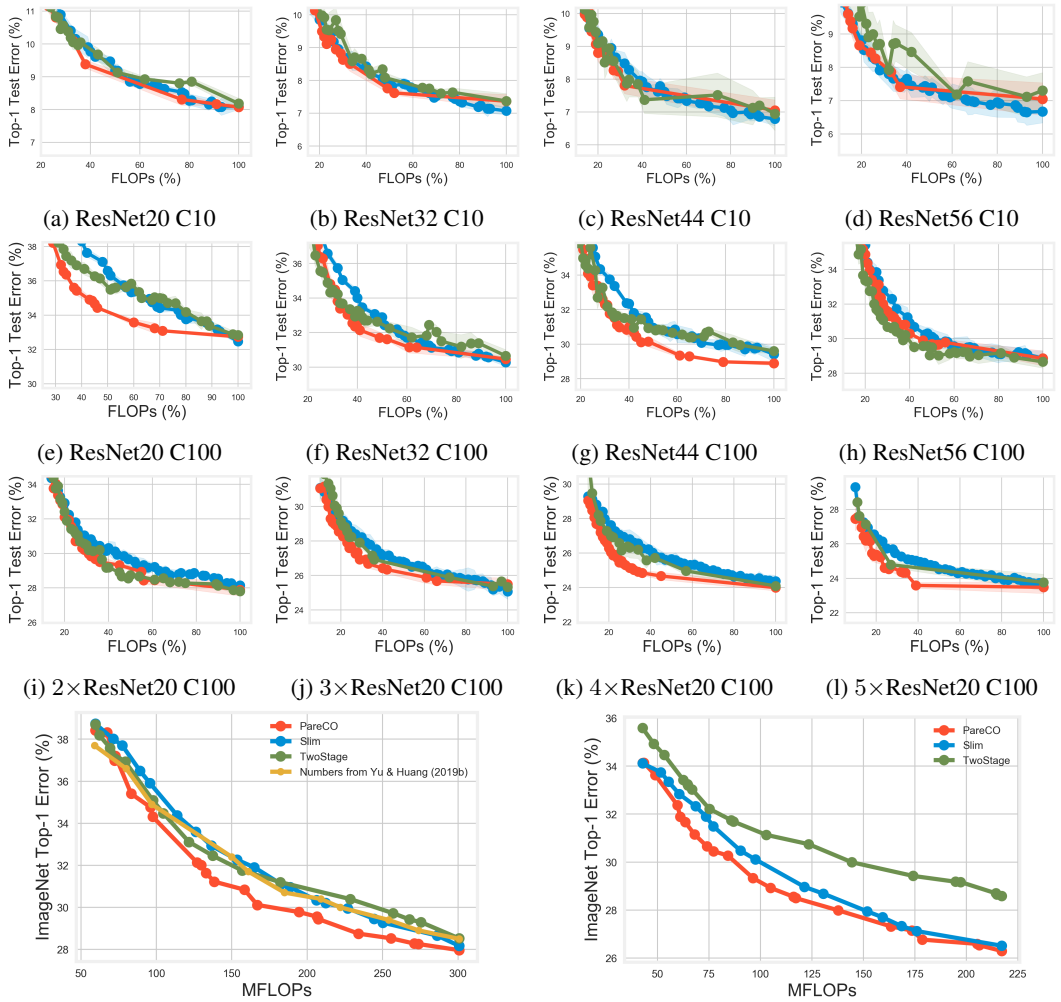
4.1 PERFORMANCE GAINS INTRODUCED BY PARECO

For all the PareCO experiments in this sub-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 (Yu & Huang, 2019b) that samples two width configurations in between the largest and the smallest widths. As for binary search, we conduct at most 10 binary searches with ϵ set to 0.02, which means that the binary search terminates if the FLOPs difference is within a two percent margin relative to the full model FLOPs. On average, the procedure terminates by using 3.4 binary searches for results on ImageNet. The dimension of $\boldsymbol{c}$ is network-dependent and is specified in Appendix B and the training hyperparameters are detailed in Appendix D. To arrive at the final set of architectures for PareCO, we use non-dominated sort based on the training loss and FLOPs for $c \in \mathcal{H}$.

We consider three datasets: CIFAR-10, CIFAR-100, and ImageNet. To provide informative comparisons, we verify our implementation for the conventional slimmable training with the reported numbers in (Yu & Huang, 2019b) using MobileNetV2 on ImageNet. Our results follow closely to the reported numbers as shown in Figure 2m, which makes our comparisons on other datasets convincing.

We compare to the following baselines:

- **Slim**: the conventional slimmable training method (the universally slimmable networks by Yu & Huang (2019b)). We select 40 architectures uniformly distributed across FLOPs and run a non-dominated sort using training loss and FLOPs to arrive at the points to be plotted.



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

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

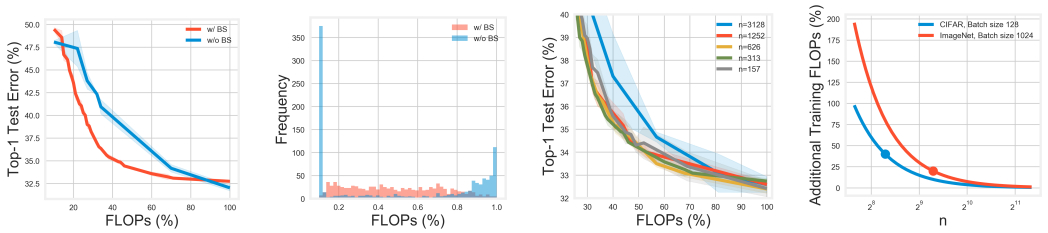
(i) 2×ResNet20 C100 (j) 3×ResNet20 C100 (k) 4×ResNet20 C100 (l) 5×ResNet20 C100

(m) MobileNetV2 ImageNet (n) MobileNetV3 ImageNet

Figure 2: Comparisons among PareCO, Slim, and TwoStage. 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. The numerical results for ImageNet are detailed in Table 1 in Appendix E.

- **TwoStage**: disjoint optimization that first trains the model with weight sharing, then uses search methods to find architectures that work well given the trained weights (similar to OFA (Cai et al., 2020) and BigNAS (Yu et al., 2020)). To compare fairly with PareCO, we use multi-objective Bayesian optimization for the search. After optimization, we run a non-dominated sort for all the visited architectures $\mathcal{H}$ using training loss and FLOPs to arrive at the points to be plotted.

Compared to *Slim*, the proposed PareCO has demonstrated much better results across various networks and datasets. This suggests that channel optimization can indeed improve the efficiency of slimmable networks. Compared to *TwoStage*, PareCO is better or comparable across networks and datasets. This suggests that training network weights by uniformly sampling architectures regardless of Pareto-efficiency can be sub-optimal. More importantly, we find that such a training techniques result in low performances for all the sub-networks in MobileNetV3, which is potentially due to conflicts in the optimization process with a large number of sub-networks and small network capacity. From the perspective of training overhead, PareCO introduced minor overhead compared to Slim due to the



(a) Impact of binary search (BS).

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

(c) Performance for different n.

(d) Additional overhead over Slim for different n.

Figure 3: Ablation study for binary search 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.

temporal similarity approximation. More specifically, on ImageNet, PareCO incurs approximately 20% extra overhead compared to Slim.

Note that the performance among these three methods are similar for the CIFAR-10 dataset. 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 CIFAR-100 and ImageNet experiments. We conjecture that this is because PareCO introduces a new optimization variable (width-multipliers), which allows better compromises to be attained.

4.2 Ablation Studies

In this subsection, 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.

Binary search Without binary search, 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 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 directly in the $\boldsymbol{\lambda}$ space results in non-uniform FLOPs and worse performance compared to binary search.

Number of samples in BO We reduce the number of samples in BO 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 Bayesian optimization. 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.

5 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

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 (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 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*, pp. 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*, pp. 1586–1595, 2018.

Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1539–1547, 2020.

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, 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*, pp. 4340–4349, 2019.

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*, pp. 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.

Yigitcan 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*, pp. 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*, pp. 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*, pp. 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*, pp. 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, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

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*, pp. 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*, pp. 11264–11272, 2019.

Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pp. 4933–4943, 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 (eds.), *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, pp. 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 *Advances in Neural Information Processing Systems 32*, pp. 8026–8037. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1gSj0NKvB`.

Adrià Ruiz and Jakob Verbeek. Adaptative inference cost with convolutional neural mixture models. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 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*, pp. 4510–4520, 2018.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

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.

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*, pp. 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.

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*, pp. 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*, pp. 7242–7251, 2019.

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

A PROOF FOR THEOREM

Let $\boldsymbol{\theta}$ be the network weights, $\mathcal{D}$ be the training data distribution, and $\boldsymbol{x}$ and y be the training input and label. Furthermore, let $\alpha(\cdot)$ be a function that maps from $\boldsymbol{\lambda}$ to the corresponding Pareto channel configuration (*i.e.*, $\alpha(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{x}} \mathcal{T}_{\boldsymbol{\lambda}}(\boldsymbol{x})$), $g(\cdot)$ be a function that maps a certain FLOPs back to the corresponding $\boldsymbol{\lambda}$, (*i.e.*, $g = (f_{\text{FLOPs}} \circ \alpha)^{-1}$), and f_{CE} be the cross entropy loss that takes channel configuration, weights and data, one can further capture the area under curve by Riemann integration over FLOPs:

$$\begin{aligned} A(\boldsymbol{\theta}, \boldsymbol{x}, y) &= \int_a^b f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z})), \boldsymbol{\theta}, \boldsymbol{x}, y) dz \\ &\approx \sum_{i=0}^{r-1} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z_i})), \boldsymbol{\theta}, \boldsymbol{x}, y)(z_{i+1} - z_i), \end{aligned} \tag{3}$$

where the approximation is done via Riemann sum with r uniform partitions over FLOPs. As a result, our formal objective for optimizing a slimmable neural network is as follows:

$$\begin{aligned} \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[A(\boldsymbol{\theta}, \boldsymbol{x}, y)] &\approx \arg\min_{\boldsymbol{\theta}} A(\boldsymbol{\theta}, \boldsymbol{x}, y) \\ &\approx \arg\min_{\boldsymbol{\theta}} \sum_{i=0}^{r-1} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z_i})), \boldsymbol{\theta}, \boldsymbol{x}, y)(z_{i+1} - z_i) \\ &= \arg\min_{\boldsymbol{\theta}} \sum_{i=0}^{r-1} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z_i})), \boldsymbol{\theta}, \boldsymbol{x}, y) \\ &= \arg\min_{\boldsymbol{\theta}} \frac{1}{r} \sum_{i=0}^{r-1} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z_i})), \boldsymbol{\theta}, \boldsymbol{x}, y) \\ &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{z\sim[a,b]} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z})), \boldsymbol{\theta}, \boldsymbol{x}, y) \\ &\approx \arg\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{m=1}^{M} f_{\text{CE}}(\alpha(\boldsymbol{g}(\boldsymbol{z^{(m)}})), \boldsymbol{\theta}, \boldsymbol{x}, y), \end{aligned} \tag{4}$$

where first and final approximation are done via Monte Carlo sampling and the second approximation is done via equation 3.

B WIDTH PARAMETERIZATION

For ResNets with CIFAR, $\boldsymbol{c}$ has six dimensions and is denoted by $\boldsymbol{c}_{1:6} \in [0.316, 1]$, *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, $\boldsymbol{c}_{1:25} \in [0.42, 1]$, 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 $\boldsymbol{c}_{1:22} \in [0.42, 1]$. 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. Note that 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 & Huang, 2019b). Due to these reasons, our $0.42\times$ MobileNetV2 has 59 MFLOPs, which has the same FLOPs as the $0.35\times$ MobileNetV2 in (Yu & Huang, 2019b; Sandler et al., 2018).

C WIDTH DIFFERENCES

In Figure 4, we visualize the widths learned by PareCO and contrast them with Slim for MobileNetV2 and MobileNetV3. Note that both PareCO and Slim are slimmable networks with shared weights and from the top row to the bottom row represent three points on the trade-off curve for Figure 2m and Figure 2n.

(a) MobileNetV2 ImageNet

(b) MobileNetV3 ImageNet

Figure 4: Comparing the width-multipliers between PareCO and Slim. The title for each plot denotes the relative differences (PareCO - Slim) and the numbers in the parenthesis are for PareCO.

D 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.

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 same exact hyperparameters but train $2\times$ longer compared to independent models, *i.e.*, 400 epochs.

ImageNet Our training hyperparameters follow that of (Yu & 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 & 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 & 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 & Huang, 2019b)[2]. The entire training is done using 8 NVIDIA V100 GPUs.

E NUMERICAL RESULTS FOR IMAGENET

We summarize the numbers from Figure 2m and Figure 2n in Table 1 for future work to compare easily.

MobileNetV2				MobileNetV3			
MFLOPs	Independently trained (Sandler et al., 2018)	Slim	PareCO	MFLOPs	Independently trained (Howard et al., 2019)	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.

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