# SLIMMABLE NETWORKS FOR CONTRASTIVE SELF-SUPERVISED LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Self-supervised learning makes great progress in large model pre-training but suffers in training small models. Previous solutions to this problem mainly rely on knowledge distillation and indeed have a two-stage learning procedure: first train a large teacher model, then distill it to improve the generalization ability of small ones. In this work, we present a new one-stage solution to obtain pre-trained small models without extra teachers: slimmable networks for contrastive self-supervised learning (*SlimCLR*). A slimmable network contains a full network and several weight-sharing sub-networks. We can pre-train for only one time and obtain various networks including small ones with low computation costs. However, in self-supervised cases, the interference between weight-sharing networks leads to severe performance degradation. One evidence of the interference is *gradient imbalance*: a small proportion of parameters produces dominant gradients during backpropagation, and the main parameters may not be fully optimized. The divergence in gradient directions of various networks may also cause interference between networks. To overcome these problems, we make the main parameters produce dominant gradients and provide consistent guidance for sub-networks via three techniques: slow start training of sub-networks, online distillation, and loss re-weighting according to model sizes. Besides, a switchable linear probe layer is applied during linear evaluation to avoid the interference of weight-sharing linear layers. We instantiate SlimCLR with typical contrastive learning frameworks and achieve better performance than previous arts with fewer parameters and FLOPs.

## 1 INTRODUCTION

In the past decade, deep learning achieves great success in different fields of artificial intelligence. A large amount of manually labeled data is the fuel behind such success. However, manually labeled data is expensive and far less than unlabeled data in practice. To relieve the constraint of costly annotations, self-supervised learning (Dosovitskiy et al., 2016; Wu et al., 2018; van den Oord et al., 2018; He et al., 2020; Chen et al., 2020a) aims to learn transferable representations for downstream tasks by training networks on unlabeled data. Great progress is made in large models, *i.e.*, models bigger than ResNet-50 (He et al., 2016) that has roughly 25M parameters. For example, ReLICv2 (Tomasev et al., 2022) achieves 77.1% accuracy on ImageNet (Russakovsky et al., 2015) under linear evaluation protocol with ResNet-50, outperforming the supervised baseline 76.5%.

In contrast to the success of the large model pre-training, self-supervised learning with small models lags behind. For instance, supervised ResNet-18 with 12M parameters achieves 72.1% accuracy on ImageNet, but its self-supervised result with MoCov2 (Chen et al., 2020c) is only 52.5% (Fang et al., 2021). The gap is nearly 20%. To fulfill the large performance gap between supervised and self-supervised small models, previous methods (Fang et al., 2021; Gao et al., 2022; Xu et al., 2022) mainly focus on knowledge distillation, namely, they try to transfer the knowledge of a self-supervised large model into small ones. Nevertheless, such methodology actually has a two-stage procedure: first train an additional large model, then train a small model to mimic the large one. Besides, one-time distillation only produces a single small model for a specific computation scenario.

An interesting question naturally arises: can we obtain different small models through one time pre-training to meet various computation scenarios without extra teachers? Inspired by the success of slimmable networks (Yu et al., 2019) in supervised learning, we present a novel one-stage

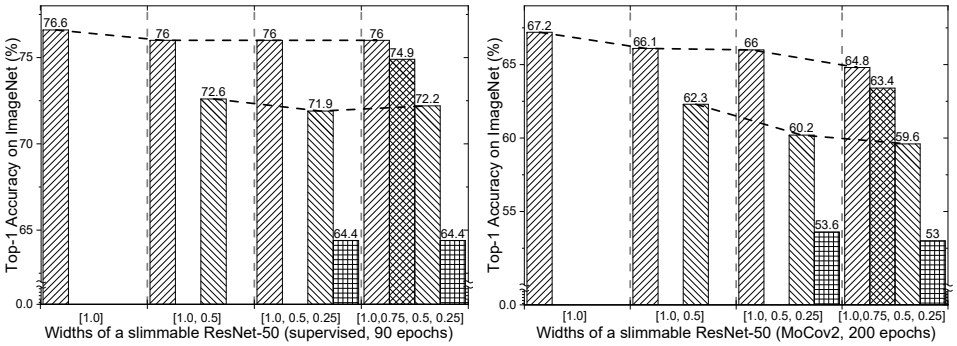



Figure 1: Training a slimmable ResNet-50 in supervised (left) and self-supervised (right) manners. ResNet-50$_{[1.0,0.75,0.5,0.25]}$ means this slimmbale network can switch at width $[1.0, 0.75, 0.5, 0.25]$. The width $0.25$ represents that the number of channels is scaled by $0.25$ of the full network.

method to obtain pre-trained small models without adding large models: slimmable networks for contrastive self-supervised learning (*SlimCLR*). A slimmable network consists of a full network and some weight-sharing sub-networks with different widths. The width denotes the number of channels in a network. Slimmable networks can execute at various widths, permitting flexible deployment on different computing devices. We can thus obtain multiple networks including small ones meeting low computing cases via one-time pre-training. Weight-sharing networks can also inherit knowledge from the large ones via the sharing parameters to achieve better generalization performance.

Weight-sharing networks in a slimmbale network cause interference to each other when training simultaneously, and the situation is worse in self-supervised cases. As shown in Figure 1, with supervision, weight-sharing networks only have a slight impact on each other, *e.g.*, the full model achieves 76.6% *vs*. 76.0% accuracy in ResNet-50$_{[1.0]}$ and ResNet-50$_{[1.0,0.75,0.5,0.25]}$. Without supervision, the corresponding numbers become 67.2% *vs*. 64.8%. One observed phenomenon of the interference is *gradient imbalance*: a small proportion of parameters produces dominant gradients during backpropagation. The imbalance occurs because the sharing parameters receive gradients from multiple losses of different networks during optimization. The main parameters may not be fully optimized due to gradient imbalance. Besides, the gradient directions of weight-sharing networks may also diverge from each other and lead to conflicts during training. More explanations and visualizations about the interference of weight-sharing networks can be found in Appendix A.3.

To relieve the gradient imbalance, the main parameters should produce dominant gradients during the optimization process. To avoid conflicts in gradient directions of various networks, sub-networks should have consistent guidance. Following these principles, we introduce three simple yet effective techniques during pre-training to relieve the interference of networks. 1) We adopt a *slow start* strategy for sub-networks. The networks and pseudo supervision of contrastive learning are both unstable and fast updating at the start of training. To avoid interference making the situation worse, we only train the full model at first. After the full model becomes relatively stable, sub-networks can inherit the knowledge via sharing parameters and start with better initialization. 2) We apply *online distillation* to make all sub-networks consistent with the full model to eliminate divergence of networks. The predictions of the full model will serve as global guidance for all sub-networks. 3) We *re-weight the losses of networks* according to their widths to ensure that the full model dominates the optimization process. Besides, we adopt a *switchable linear probe layer* to avoid interference of weight-sharing linear layers during evaluation. A single slimmable linear layer cannot achieve several complex mappings simultaneously when the data distribution is complicated.

We instantiate two algorithms for SlimCLR with typical contrastive learning frameworks, *i.e.*, MoCov2 and MoCov3 (Chen et al., 2020c; 2021). Extensive experiments are done on ImageNet (Russakovsky et al., 2015) dataset, and the results show that our methods achieve significant performance improvements compared to previous arts with fewer parameters and FLOPs.

## 2 Related Works

**Self-supervised learning** Self-supervised learning aims to learn transferable representations for downstream tasks from input data itself. According to Liu et al. (2020), self-supervised methods

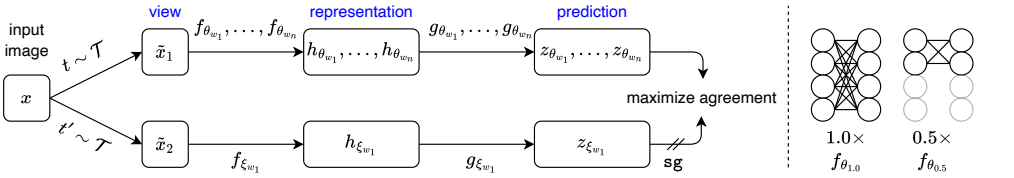
can be summarized into three main categories according to their objectives: *generative*, *contrastive*, and *generative-contrastive (adversarial)*. Methods belonging to the same categories can be further classified by the difference between pretext tasks. Given input $x$, generative methods encode $x$ into an explicit vector $z$ and decode $z$ to reconstruct $x$ from $z$, *e.g.*, auto-regressive (van den Oord et al., 2016a;b), auto-encoding models (Ballard, 1987; Kingma & Welling, 2014; Devlin et al., 2019; He et al., 2022). Contrastive learning methods encoder input $x$ into an explicit vector $z$ to measure similarity. The two mainstream methods below this category are context-instance contrast (info-Max Hjelm et al. (2019), CPC van den Oord et al. (2018), AMDIM Bachman et al. (2019)) and instance-instance contrast (DeepCluster Caron et al. (2018), MoCo He et al. (2020); Chen et al. (2021), SimCLR Chen et al. (2020a;b), SimSiam Chen & He (2021)). Generative-contrastive methods generate a fake sample $x'$ from $x$ and try to distinguish $x'$ from real samples, *e.g.*, DCGANs Radford et al. (2016), inpainting Pathak et al. (2016), and colorization Zhang et al. (2016).

**Slimmable neworks** Slimmable networks are first proposed to achieve instant and adaptive accuracy-efficiency trade-offs on different devices (Yu et al., 2019). It can execute at different widths during runtime. Following the pioneering work, universally slimmable networks (Yu & Huang, 2019b) develop systematic training approaches to allow slimmable networks to run at arbitrary widths. AutoSlim (Yu & Huang, 2019a) further achieves one-shot architecture search for channel numbers under a certain computation budget. MutualNet (Yang et al., 2020) trains slimmable networks using different input resolutions to learn multi-scale representations. Dynamic slimmable networks (Li et al., 2022; 2021) change the number of channels of each layer in the fly according to the input. In contrast to weight-sharing sub-networks in slimmable networks, some methods try to train multiple sub-networks with independent parameters (Zhao et al., 2022b). A relevant concept of slimmable networks in network pruning is *network slimming* (Liu et al., 2017; Chavan et al., 2022; Wang et al., 2021), which aims to achieve channel-level sparsity for computation efficiency.

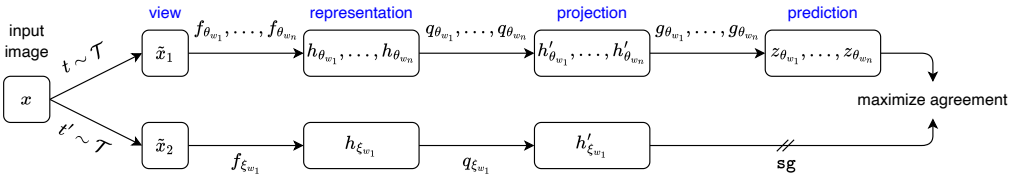
# 3 METHOD

## 3.1 DESCRIPTION OF SLIMCLR

We develop two instantial algorithms for SlimCLR with typical contrastive learning frameworks MoCov2 and MoCov3 (Chen et al., 2020c; 2021). As shown in Figure 2a (right), a slimmable network with $n$ widths $w_1, \ldots, w_n$ contains multiple weight-sharing networks $f_{\theta_{w_1}}, \ldots, f_{\theta_{w_n}}$, which are parameterized by learnable weights $\theta_{w_1}, \ldots, \theta_{w_n}$, respectively. Each network $f_{\theta_{w_i}}$ in the slimmable network has its own set of weights $\Theta_{w_i}$ and $\theta_{w_i} \in \Theta_{w_i}$. A network with a small width shares the weights with large ones, namely, $\Theta_{w_j} \subset \Theta_{w_i}$ if $w_j < w_i$. Generally, we assume $w_j < w_i$ if $j > i$, *i.e.*, $w_1, \ldots, w_n$ arrange in descending order, and $\theta_{w_1}$ represent the parameters of the full model.




(a) left: SlimCLR-MoCov2. right: a slimmable network with two widths $[1.0, 0.5]$.




(b) SlimCLR-MoCov3

Figure 2: The overall framework of SlimCLR. A slimmable network produces different outputs from weight-sharing networks with various widths $w_1, \ldots, w_n$, where $w_1$ is the width of the full model. $\theta$ are the network parameters and $\xi$ are an exponential moving average of $\theta$. sg means stop-gradient.

We first illustrate the learning process of SlimCLR-MoCov2 in Figure 2a. Given a set of images $\mathcal{D}$, an image $x$ sampled uniformly from $\mathcal{D}$, and one distribution of image augmentation $\mathcal{T}$, SlimCLR produces two data views $\tilde{x}_1 = t(x)$ and $\tilde{x}_2 = t'(x)$ from $x$ by applying augmentations $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$, respectively. For the first view, SlimCLR outputs multiple representations $h_{\theta_{w_1}}, \ldots, h_{\theta_{w_n}}$ and predictions $z_{\theta_{w_1}}, \ldots, z_{\theta_{w_n}}$, where $h_{\theta_{w_i}} = f_{\theta_{w_i}}(\tilde{x}_1)$ and $z_{\theta_{w_i}} = g_{\theta_{w_i}}(h_{\theta_{w_i}})$. $g$ is a stack of slimmable linear transformation layers, *i.e.*, a slimmable version of the MLP head in MoCov2 and SimCLR (Chen et al., 2020a). For the second view, SlimCLR only outputs a single representation from the full model $h_{\xi_{w_1}} = f_{\xi_{w_1}}(\tilde{x}_2)$ and prediction $z_{\xi_{w_1}} = g_{\xi_{w_1}}(h_{\xi_{w_1}})$. We minimize the InfoNCE (van den Oord et al., 2018) loss to maximize the similarity of positive pairs $z_{\theta_{w_i}}$ and $z_{\xi_{w_1}}$:

$$\mathcal{L}_{z_{\theta_{w_i}}, z_\xi, \{z^-\}} = -\log \frac{\exp(\overline{z}_{\theta_{w_i}} \cdot \overline{z}_{\xi_{w_1}}/\tau_1)}{\exp(\overline{z}_{\theta_{w_i}} \cdot \overline{z}_{\xi_{w_1}}/\tau_1) + \sum_{z^-} \exp(\overline{z}_{\theta_{w_i}} \cdot z^-/\tau_1)}, \tag{1}$$

where $\overline{z}_{\theta_{w_i}} = z_{\theta_{w_i}}/\|z_{\theta_{w_i}}\|_2$, $\overline{z}_{\xi_{w_1}} = z_{\xi_{w_1}}/\|z_{\xi_{w_1}}\|_2$, $\tau_1$ is a temperature hyper-parameter, and $\{z^-\}$ are features of negative samples. For SlimCLR-MoCov2, $\{z^-\}$ comes from a queue. Following MoCov2, the queue is updated by $\overline{z}_{\xi_{w_1}}$ every iteration during training. The overall objective is the sum of losses of all networks with various widths:

$$\mathcal{L}_{z_\theta, z_\xi, \{z^-\}} = \sum_{i=1}^{n} \mathcal{L}_{z_{\theta_{w_i}}, z_\xi, \{z^-\}}. \tag{2}$$

$\xi$ is updated by $\theta$ every iteration: $\xi \leftarrow m\xi + (1-m)\theta$, where $m \in [0, 1)$ is a momemtum coefficient.

Compared to SlimCLR-MoCov2, SlimCLR-MoCov3 has an additional projection process. It first projects the representation to another high dimensional space, then makes predictions. The projector $q$ is a stack of slimmable linear transformation layers. SlimCLR-MoCov3 also adopts the InfoNCE loss, but the negative samples come from other samples in the mini-batch.

After contrastive learning, we only keep $f_{\theta_{w_1}}, \ldots, f_{\theta_{w_n}}$ and abandon other components.

### 3.2 Gradient imbalance and solutions

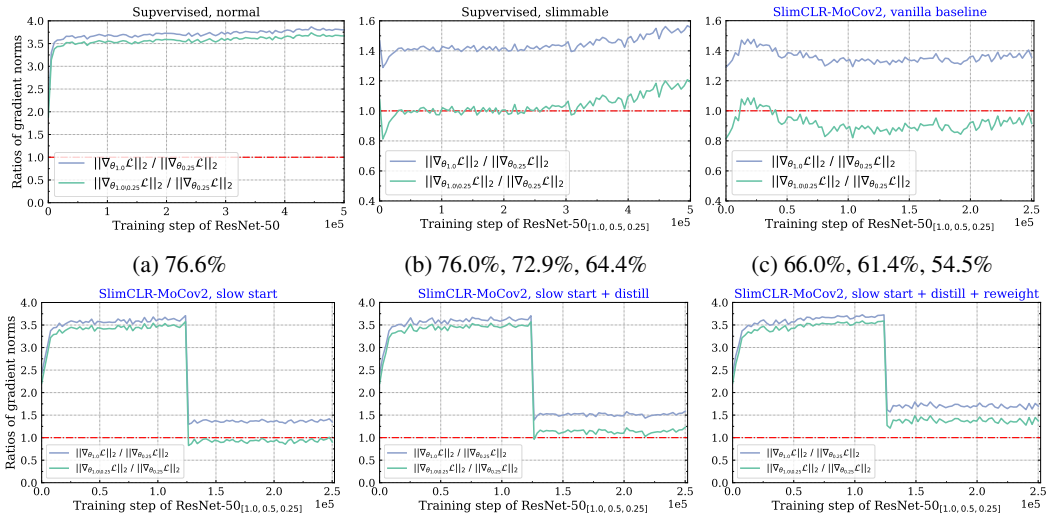
As shown in Figure 1, a vanilla implementation of the above framework leads to server performance degradation as weight-sharing networks interfere with each other during pre-training. One evidence of such interference we observed is *gradient imbalance*.

*Gradient imbalance* refers to that a small proportion of parameters produces dominant gradients during backpropagation. To quantitatively evaluate the phenomenon, we show the ratios of gradient norms of main and minor parameters: $\|\nabla_{\theta_{1.0}}\mathcal{L}\|_2$ and $\|\nabla_{\theta_{1.0\backslash 0.25}}\mathcal{L}\|_2$ versus $\|\nabla_{\theta_{0.25}}\mathcal{L}\|_2$ in Figure 3, where $\mathcal{L}$ is the loss function. Meanwhile, the ratio of the numbers of parameters is $\frac{|\Theta_{1.0}\backslash\Theta_{0.25}|}{|\Theta_{0.25}|} \approx 15$, where $\theta_{1.0\backslash 0.25} \in \Theta_{1.0}\backslash\Theta_{0.25}$. Generally, the main parameters dominate the optimization process and produce large gradient norms, *i.e.*, the two ratios should both be large ($> 1$). In Figure 3a, the two ratios are both around 3.5 when training a normal network. However, in Figure 3b and 3c, when training a slimmable network, gradient imbalance occurs because sharing parameters obtain multiple gradients from different losses. To be specific, if the widths of a slimmable network $w_1, \ldots, w_n$ arrange in a descending order and the training loss is $\mathcal{L}_{z_\theta, z_\xi, \{z^-\}}$, $\theta_{w_n}$ that only represent a small part of parameters will receive gradients from $n$ different losses and obtain a large gradient norm:

$$\nabla_{\theta_{w_n}} \mathcal{L}_{z_\theta, z_\xi, \{z^-\}} = \frac{\partial \mathcal{L}_{z_\theta, z_\xi, \{z^-\}}}{\partial \theta_{w_n}} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}_{z_{\theta_{w_i}}, z_\xi, \{z^-\}}}{\partial \theta_{w_n}}. \tag{3}$$

Gradient imbalance is more obvious in self-supervised cases. In the supervised case in Figure 3b, $\|\nabla_{\theta_{1.0\backslash 0.25}}\mathcal{L}\|_2$ is close to $\|\nabla_{\theta_{0.25}}\mathcal{L}\|_2$ at first, and the former becomes larger along with the training process. By contrast, for vanilla SlimCLR-MoCov2 in Figure 3c, $\|\nabla_{\theta_{1.0\backslash 0.25}}\mathcal{L}\|_2$ is smaller than the other most time. A conjecture is that instance discrimination is harder than supervised classification. Consequently, small networks with limited capacity are hard to convergence, produce large losses, and cause more disturbances to other weight-sharing networks.

When gradient imbalance occurs, the main parameters may not be fully optimized. Besides, the gradient directions of weight-sharing networks may also diverge from each other during backpropagation. All these lead to interference between weight-sharing networks and performance degradation.




(a) 76.6% (b) 76.0%, 72.9%, 64.4% (c) 66.0%, 61.4%, 54.5%

(d) 66.3%, 62.2%, 55.1% (e) 66.9%, 62.3%, 54.9% (f) 67.0%, 62.4%, 55.1%

Figure 3: Ratios of gradient norms: $\frac{\|\nabla_{\theta_{1.0}}\mathcal{L}\|_2}{\|\nabla_{\theta_{0.25}}\mathcal{L}\|_2}$ and $\frac{\|\nabla_{\theta_{1.0\setminus 0.25}}\mathcal{L}\|_2}{\|\nabla_{\theta_{0.25}}\mathcal{L}\|_2}$. The gradient norm of the network is calculated by averaging the layer-wise $\ell_2$ gradient norms. $\nabla_{\theta_{1.0\setminus 0.25}}\mathcal{L}$ is the gradient of the final loss *w.r.t.* parameters $\theta_{1.0\setminus 0.25} \in \Theta_{1.0}\setminus\Theta_{0.25}$, *i.e.*, rest parameters of $\Theta_{1.0}$ besides $\Theta_{0.25}$.

To avoid gradient imbalance, one natural idea is to make the main parameters dominate the optimization process, *i.e.*, the two ratios in Figure 3 should both be large. To resolve the possible conflicts of gradient directions, networks should have a consistent optimization goal. In order to achieve the above purposes, we develop three simple yet effective techniques during pre-training: *slow start*, *online distillation*, and *loss reweighting*. Besides, we further introduce a *switchable linear probe layer* to avoid the interference of weight-sharing linear layers during linear evaluation.

**slow start** At the start of training, the model and pseudo supervision of contrastive learning are both fast updating. The optimization procedure is unstable. To avoid interference between weight-sharing networks making the situation harder, at the first $S$ epochs, we only train the full model, *i.e.*, only update $\theta_{1.0}$ by $\nabla_{\theta_{1.0}}\mathcal{L}_{z_{\theta_{1.0}}, z_\xi, \{z^-\}}$. In Figure 3d, the ratios of gradient norms are large before the $S$-th epoch; then they dramatically become small after slow start. At the first $S$ epochs, the full model can learn certain knowledge from the data without disturbances, and sub-networks can inherit the knowledge via the sharing parameters and start with better initialization at the $S$-th epoch.

**online distillation** The full model has the greatest capacity to learn knowledge from the data. The prediction of the full model can serve as consistent guidance for all sub-networks to resolve the conflicts of weight-sharing networks. Following Yu & Huang (2019b), we minimize the Kullback-Leibler (KL) divergence between the estimated probabilities $p_{w_i} = \frac{\exp(\overline{z}_{\theta_{w_i}} \cdot \overline{z}_{\xi_{w_1}}/\tau_2)}{\exp(\overline{z}_{\theta_{w_i}} \cdot \overline{z}_{\xi_{w_1}}/\tau_2) + \sum_{z^-}\exp(\overline{z}_{\theta_{w_i}} \cdot z^-/\tau_2)}$ of sub-networks and the full model:

$$\mathcal{L}_{p_{w_i}} = -p_{w_1} \log p_{w_i} \quad \text{where} \quad w_i \in \{w_2, \ldots, w_n\}. \tag{4}$$

$\tau_2$ is a temperature coefficient of distillation. In Figure 3e, we observe that online distillation helps $\|\nabla_{\theta_{1.0\setminus 0.25}}\mathcal{L}\|_2 / \|\nabla_{\theta_{0.25}}\mathcal{L}\|_2$ become larger than 1.0. This means that online distillation also relieves the gradient imbalance and helps the main parameters dominate the optimization process.

**loss reweighting** Another straightforward solution to gradient imbalance is to assign large confidence to networks with large widths. We adopt a strategy in which the strongest takes control. The weight for the loss of the network with width $w_i$ is:

$$\lambda_i = 1.0 + \mathbf{1}\{w_i = w_1\} \times \sum_{j=2}^{n} w_j, \tag{5}$$

where $\mathbf{1}\{\cdot\}$ equals to 1 if the inner condition is true, 0 otherwise. In Figure 3f, both ratios become large, and $\|\nabla_{\theta_{1.0\setminus 0.25}}\mathcal{L}\|_2/\|\nabla_{\theta_{0.25}}\mathcal{L}\|_2$ are larger than 1.0 by a clear margin. Loss reweighting helps the main parameters produce large gradient norms and dominate the optimization process.

The overall pre-training objective of SlimCLR is:

$$\mathcal{L}_{all} = \lambda_1 \mathcal{L}_{z_{\theta_{w_1}}, z_\xi, \{z^-\}} + \sum_{i=2}^{n} \lambda_i \frac{(\mathcal{L}_{z_{\theta_{w_i}}, z_\xi, \{z^-\}} + \mathcal{L}_{p_{w_i}})}{2}. \tag{6}$$

**switchable linear probe layer** After pre-training, we observe that a single slimmable linear layer is not able to achieve several complex mappings from different representations to the same object classes simultaneously. We provide theoretical results about the conditions of inputs when using a single slimmable linear layer to solve multiple multi-class linear regression problems in Appendix A.1. The failure of a single slimmable linear layer is possibly because the learned representations in Figure 2 do not meet the requirement. In this case, we propose a switchable linear probe layer mechanism. Namely, each network in the slimmable network will have its own linear probe layer for linear evaluation. We also apply the online distillation strategy during fine-tuning.

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL DETAILS

**Datatest** We train SlimCLR on ImageNet (Russakovsky et al., 2015), which contains 1.28M training and 50K validation images. During pre-training, we use training images without labels.

**Learning strategies of SlimCLR-MoCov2** By default, we use a total batch size 1024, an initial learning rate 0.2, and weight decay $1 \times 10^{-4}$. We adopt the SGD optimizer with a momentum 0.9. A linear warm-up and cosine decay policy (Goyal et al., 2017; He et al., 2019) for learning rate is applied, and the warm-up epoch is 10. The temperatures are $\tau_1 = 0.2$ for InfoNCE and $\tau_2 = 5.0$ for online distillation. Without special mentions, other settings including data augmentations, queue size (65536), and feature dimension (128) are the same as the counterparts of MoCov2 (Chen et al., 2020c). The slow start epoch $S$ of sub-networks is set to be half of the number of total epochs.

**Learning strategies of SlimCLR-MoCov3** We use a total batch size 1024, an initial learning rate 1.2, and weight decay $1 \times 10^{-6}$. We adopt the LARS (You et al., 2017) optimizer and a cosine learning rate policy with warm-up epoch 10. The temperatures are $\tau_1 = 1.0$ and $\tau_2 = 1.0$. The slow start epoch $S$ is half of the total epochs. One different thing is that we increase the initial learning rate to 3.2 after $S$ epochs. Pre-training is all done with mixed precision (Micikevicius et al., 2018).

**Linear evaluation** Following the general linear evaluation protocol (Chen et al., 2020a; He et al., 2020), we add new linear transformation layers on the backbone and freeze the parameters of the backbone during evaluation. As described above, we also apply online distillation with a temperature $\tau_2 = 1.0$ when training these linear layers. For linear evaluation of SlimCLR-MoCov2, we use a total batch size 1024, total epochs 100, and an initial learning rate 60, which is decayed by 10 at 60 and 80 epochs. For linear evaluation of SlimCLR-MoCov3, we use a total batch size 1024, total epochs 90, and an initial learning rate 0.4 with cosine decay policy.

## 4.2 RESULTS OF SLIMCLR ON IMAGENET

Results of SlimCLR on ImageNet are shown in Table 1. Even though we pay huge efforts to relieve the interference of weight-sharing networks as described in Section 3.2, slimmable training inevitably leads to a drop in performance for the full model. When training for more epochs, the degradation is more obvious. However, such degradation also occurs in the supervised case. Considering the advantages of slimmable training we will discuss below, the degradation is acceptable.

Compared to MoCov2 with individual networks, SlimCLR helps sub-networks achieve significant performance improvements. Specifically, for ResNet-$50_{0.5}$ and ResNet-$50_{0.25}$, SlimCLR-MoCov2 achieves 3.5% and 6.6% improvements in performance when pre-training for 200 epochs, respectively. This verifies that sub-networks can inherit knowledge from the full model via sharing parameters to improve their generalization ability. We can also use more powerful contrastive learning framework to further boost the performance of sub-networks, *i.e.*, SlimCLR-MoCov3.

| Method | Backbone | Teacher | Top-1 | Top-5 | Epochs | #Params | #FLOPs |
|---|---|---|---|---|---|---|---|
| Supervised | R-50 | ✗ | 76.6 | 93.2 | 100 | 25.6 M | 4.1 G |
| | R-34 | | 75.0 | - | - | 21.8 M | 3.7 G |
| | R-18 | | 72.1 | - | - | 11.9 M | 1.8 G |
| | R-50$_{1.0}$ | ✗ | 76.0$_{(0.6\downarrow)}$ | 92.9 | 100 | 25.6 M | 4.1 G |
| | R-50$_{0.75}$ | | 74.9 | 92.3 | | 14.7 M | 2.3 G |
| | R-50$_{0.5}$ | | 72.2 | 90.8 | | 6.9 M | 1.1 G |
| | R-50$_{0.25}$ | | 64.4 | 86.0 | | 2.0 M | 278 M |
| Baseline (individual networks trained with MoCov2) | R-50 | ✗ | 67.5 | - | 200 | 25.6 M | 4.1 G |
| | R-50$_{1.0}$ | ✗ | 67.2 | 87.8 | 200 | 25.6 M | 4.1 G |
| | R-50$_{0.75}$ | | 64.3 | 85.8 | | 14.7 M | 2.3 G |
| | R-50$_{0.5}$ | | 58.9 | 82.2 | | 6.9 M | 1.1 G |
| | R-50$_{0.25}$ | | 47.9 | 72.8 | | 2.0 M | 278 M |
| MoCov2 (2020c, preprint) | R-50 | ✗ | 71.1 | - | 800 | 25.6 M | 4.1 G |
| MoCov3 (2021, ICCV) | R-50 | | 72.8 | - | 300 | | |
| SlimCLR-MoCov2 | R-50$_{1.0}$ | | 67.4$_{(0.1\downarrow)}$ | 87.9 | 200 | | |
| SlimCLR-MoCov2 | R-50$_{1.0}$ | | 70.4$_{(0.7\downarrow)}$ | 89.6 | 800 | | |
| SlimCLR-MoCov3 | R-50$_{1.0}$ | | 72.3$_{(0.5\downarrow)}$ | 90.8 | 300 | | |
| SEED (2021, ICLR) | R-34 | R-50 (67.4) | 58.5 | 82.6 | 200 | 21.8 M | 3.7 G |
| DisCo (2022, ECCV) | R-34 | R-50 (67.4) | 62.5 | 85.4 | 200 | | |
| BINGO (2022, ICLR) | R-34 | R-50 (67.4) | 63.5 | 85.7 | 200 | | |
| SEED (2021, ICLR) | R-34 | R-50×2 (77.3) | 65.7 | 86.8 | 800 | | |
| DisCo (2022, ECCV) | R-34 | R-50×2 (77.3) | 67.6 | 88.6 | 200 | | |
| BINGO (2022, ICLR) | R-34 | R-50×2 (77.3) | 68.9$_{(6.1)}$ | 88.9 | 200 | | |
| SlimCLR-MoCov2 | R-50$_{0.75}$ | ✗ | 65.5 | 87.0 | 200 | 14.7 M | 2.3 G |
| SlimCLR-MoCov2 | R-50$_{0.75}$ | | 68.8 | 88.8 | 800 | | |
| SlimCLR-MoCov3 | R-50$_{0.75}$ | | **69.7**$_{(5.2)}$ | **89.4** | 300 | | |
| CompRess (2020, NeurIPS) | R-18 | R-50 (71.1) | 62.6 | - | 130 | 11.9 M | 1.8 G |
| SEED (2021, ICLR) | R-18 | R-50×2 (77.3) | 63.0 | 84.9 | 800 | | |
| DisCo (2022, ECCV) | R-18 | R-50×2 (77.3) | 65.2 | 86.8 | 200 | | |
| BINGO (2022, ICLR) | R-18 | R-50×2 (77.3) | 65.5 | 87.0 | 200 | | |
| SEED (2021, ICLR) | R-18 | R-152 (74.1) | 59.5 | 65.5 | 200 | | |
| DisCo (2022, ECCV) | R-18 | R-152 (74.1) | 65.5 | 86.7 | 200 | | |
| BINGO (2022, ICLR) | R-18 | R-152 (74.1) | 65.9$_{(6.2)}$ | 87.1 | 200 | | |
| SlimCLR-MoCov2 | R-50$_{0.5}$ | ✗ | 62.5 | 84.8 | 200 | 6.9 M | 1.1 G |
| SlimCLR-MoCov2 | R-50$_{0.5}$ | | 65.6 | 87.2 | 800 | | |
| SlimCLR-MoCov3 | R-50$_{0.5}$ | | **67.6**$_{(4.6)}$ | **88.2** | 300 | | |
| SlimCLR-MoCov2 | R-50$_{0.25}$ | ✗ | 55.1 | 79.5 | 200 | 2.0 M | 278 M |
| SlimCLR-MoCov2 | R-50$_{0.25}$ | | 57.6 | 81.5 | 800 | | |
| SlimCLR-MoCov3 | R-50$_{0.25}$ | | **62.4**$_{(2.0)}$ | **84.4** | 300 | | |

Table 1: Linear evaluation results of SlimCLR with ResNet-50$_{[1.0,0.75,0.5.0.25]}$ on ImageNet. Through only one-time pre-training, SlimCLR obtains multiple different small models without extra large teacher models. It also outperforms previous methods using ResNet as backbones. The performance degradation when training a slimmable network is shown in cyan. The gaps between the self-supervised and supervised results are shown in orange. The smaller, the better.

Compared to previous methods that aim to distill the knowledge of large teacher models, sub-networks of ResNet-50$_{[1.0,0.75,0.5.0.25]}$ achieve better performance with fewer parameters and FLOPs. SlimCLR also helps small models get closer performance to their supervised counterparts. Furthermore, SlimCLR does not need any additional training process of large teacher models, and all networks in SlimCLR are trained jointly. By only training for one time, we can get different models with various computation cost which are suitable for different devices. This demonstrates the superiority of adopting slimmable networks for contrastive learning to get pre-trained small models.

### 4.3 Discussion

In this section, we will discuss the influences of different components in SlimCLR.

**switchable linear probe layer** The influence of the switchable linear probe layer is shown in Table 2a. A switchable linear probe layer brings significant improvements in accuracy compared to slimmable linear probe layer. For only one slimmable layer, the interference between weight-sharing

| Model | slimmable | | switchable | |
|---|---|---|---|---|
| | Top-1 | Top-5 | Top-1 | Top-5 |
| R-50$_{1.0}$ | 64.8 | 86.1 | **65.6** | **86.8** |
| R-50$_{0.75}$ | 63.4 | 85.3 | **64.3** | **86.0** |
| R-50$_{0.5}$ | 59.6 | 82.9 | **61.3** | **84.1** |
| R-50$_{0.25}$ | 53.0 | 77.8 | **54.5** | **79.1** |

(a) switchable linear probe layer

| Model | $S = 0$ | | $S = 100$ | |
|---|---|---|---|---|
| | Top-1 | Top-5 | Top-1 | Top-5 |
| R-50$_{1.0}$ | 65.6 | 86.8 | **66.7** | **87.5** |
| R-50$_{0.75}$ | 64.3 | 86.0 | **65.3** | **86.4** |
| R-50$_{0.5}$ | 61.3 | 84.1 | **62.5** | **84.3** |
| R-50$_{0.25}$ | 54.5 | 79.1 | **54.9** | **79.5** |

(b) slow start epoch $S$, 200 epochs

| Model | MSE | ATKD | DKD | KD |
|---|---|---|---|---|
| | Top-1 | | | |
| R-50$_{1.0}$ | 66.9 | 66.4 | 66.8 | **67.0** |
| R-50$_{0.75}$ | 65.2 | 65.0 | 65.1 | **65.3** |
| R-50$_{0.5}$ | 62.4 | 62.2 | 62.3 | **62.6** |
| R-50$_{0.25}$ | **55.2** | 54.7 | 54.9 | 54.9 |

(c) online distillation loss choice

| Model | 3.0 | 4.0 | 5.0 | 6.0 |
|---|---|---|---|---|
| | Top-1 | | | |
| R-50$_{1.0}$ | **67.0** | **67.0** | **67.0** | 66.7 |
| R-50$_{0.75}$ | 65.2 | **65.3** | **65.3** | 65.2 |
| R-50$_{0.5}$ | 62.4 | **62.6** | **62.6** | 62.4 |
| R-50$_{0.25}$ | **55.0** | 54.8 | 54.9 | **55.0** |

(d) online distillation temperature

| Model | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| | Top-1 | | | |
| R-50$_{1.0}$ | 67.4 | 67.3 | **67.5** | **67.5** |
| R-50$_{0.75}$ | 65.5 | 65.7 | **65.9** | 65.8 |
| R-50$_{0.5}$ | 62.5 | 62.2 | **62.6** | 62.4 |
| R-50$_{0.25}$ | **55.1** | 54.5 | 54.4 | 54.5 |

(e) loss reweighting

| Model | 200 | 300 | 400 | 500 |
|---|---|---|---|---|
| | Top-1 | | | |
| R-50$_{1.0}$ | 70.2 | 70.0 | **70.3** | 70.1 |
| R-50$_{0.75}$ | 68.3 | 68.6 | **68.8** | 68.4 |
| R-50$_{0.5}$ | **65.7** | 65.6 | 65.6 | 65.3 |
| R-50$_{0.25}$ | **57.8** | 57.5 | 57.6 | 57.3 |

(f) slow start epoch $S$, 800 epochs

Table 2: Ablation experiments with SlimCLR-MoCov2 on ImageNet. The experiment in a former table serves as a baseline for the consequent table.

linear layers is not unavoidable. It is also possible that the learned representations of pre-trained models do not meet the requirements as we discussed in Appendix A.1.

**slow start and training time** Experiments with and without slow start are shown in Table 2b. The pre-training time of SlimCLR-MoCov2 without and with slow start epoch $S = 100$ on 8 Tesla V100 GPUs are 45 and 33 hours, respectively. For reference, the pre-training time of MoCov2 with ResNet-50 is roughly 20 hours. Slow start largely reduces the pre-training time. It also avoids the interference between weight-sharing networks at the start stage of training and helps the system reach a steady point fast during optimization. Thus sub-networks can start with good initialization and achieve better performance. We also provide ablations of slow start epoch $S$ when training for a longer time in Table 2f. Setting $S$ to be half of the total epochs is a natural and appropriate choice.

**online distillation** Here we compare two classical distillation losses: mean-square-error (MSE) and KL divergence (KD), and two other distillation losses from recent works: ATKD (Guo, 2022) and DKD (Zhao et al., 2022a). ATKD reduces the difference in sharpness between distributions of teacher and student to help the student better mimic the teacher. DKD decouples the classical knowledge distillation objective function (Hinton et al., 2015) into target class and non-target class knowledge distillation to achieve more effective and flexible distillation. In Table 2c, we can see that these four distillation losses make trivial differences in our context.

Combining the results of distillation with ResNet$_{[1.0,0.5,0.25]}$ in Figure 3e, we find that distillation mainly improves the performance of the full model, and the improvements of sub-networks are relatively small. This violates the purpose of knowledge distillation: distill the knowledge of large models and improve the performance of small ones. This is possibly because sub-networks in a slimmable network already inherit the knowledge from the full model via the sharing weights, and feature distillation cannot help the sub-networks much in this case. The main function of online distillation in our context is to relieve the interference of sub-networks as shown in Figure 3e.

We also test the influence of different temperatures in online distillation, *i.e.*, $\tau_2$ related to $p_{w_i}$ in equation 4, and results are shown in Table 2d. Following classical KD (Hinton et al., 2015), we choose $\tau_2 \in \{3.0, 4.0, 5.0, 6.0\}$. The choices of temperatures make a trivial difference in our context. SEED (Fang et al., 2021) use a small temperature 0.01 for the teacher to get sharp distribution and a temperature 0.2 for the student. BINGO (Xu et al., 2022) adopts a single temperature 0.2. Their choices are quite different from ours, and SlimCLR is more robust to the choice of temperatures. We further provide an analysis of the influence of temperatures in Appendix A.2.

**loss reweighting** We compared four loss reweighting manners in Table 2e. They are

$$(1).\ \lambda_i = 1.0 + \mathbf{1}\{w_i = w_1\} \times \sum_{j=2}^{n} w_j, \quad (2).\ \lambda_i = 1.0 + \mathbf{1}\{w_i = w_1\} \times \max\{w_2, \dots, w_n\},$$
$$(3).\ \lambda_i = n \times \frac{w_i}{\sum_{j=1}^{n} w_j}, \quad (4).\ \lambda_i = n \times \frac{1.0 + \sum_{j=i}^{n} w_j}{\sum_{j=1}^{n}(1.0 + \sum_{k=j}^{n} w_k)},$$

where $\mathbf{1}\{\cdot\}$ equals to 1 if the inner condition is true, 0 otherwise. The corresponding weights of ResNet-50$_{[1.0,0.75,0.5,0.25]}$ are $[2.5, 1.0, 1.0, 1.0]$, $[1.75, 1.0, 1.0, 1.0]$, $[1.6, 1.2, 0.8, 0.4]$, and $[1.54, 1.08, 0.77, 0.62]$. It is clear that a larger weight for the full model helps the system achieve

| Model | $\tau_2 = 5.0$ | | $\tau_2 = 1.0$ | |
|---|---|---|---|---|
| | Top-1 | Top-5 | Top-1 | Top-5 |
| R-50$_{1.0}$ | 69.0 | 89.0 | **69.2** | **89.1** |
| R-50$_{0.75}$ | 64.0 | 86.0 | **64.5** | **86.3** |
| R-50$_{0.5}$ | 62.5 | 84.8 | **62.8** | **85.0** |
| R-50$_{0.25}$ | 57.2 | 80.8 | **57.2** | **80.9** |

(a) temperature $\tau_2$, 100 epochs

| Model | 1.2 | 2.8 | 3.2 | 3.6 |
|---|---|---|---|---|
| | Top-1 | | | |
| R-50$_{1.0}$ | 70.7 | 71.9 | **72.3** | 72.1 |
| R-50$_{0.75}$ | 66.8 | 69.4 | 69.6 | **69.8** |
| R-50$_{0.5}$ | 64.7 | 67.2 | 67.6 | **68.0** |
| R-50$_{0.25}$ | 59.1 | 61.8 | 62.1 | **62.4** |

(b) $lr$ for slow start, 300 epochs

Table 3: Ablation experiments with SlimCLR-MoCov3 on ImageNet.

better performance. This demonstrates again that it is important for the full model to lead the optimization direction during training. The differences of the above four loss reweighting strategies are mainly reflected in the sub-networks with small sizes. To ensure the performance of the smallest network, we adopt the reweighting manner (1) in practice.

**ablation study with SlimCLR-MoCov3** Besides the above ablation studies with SlimCLR-MoCov2, we also provide empirical analysis for SlimCLR-MoCov3. Different from SlimCLR-MoCov2, SlimCLR-MoCov3 adopts different temperatures $\tau_1 = 1.0$ and $\tau_2 = 1.0$. When applying slow start, SlimCLR-MoCov3 also increases the initial learning rate at the same time.

In Table 3a, we show the influence of temperature $\tau_2$ for online distillation. Different from the choice in SlimCLR (MoCov2), it is better for SlimCLR-MoCov3 to choose a temperature $\tau_2$ for online distillation, which is close to the temperature $\tau_1$ of contrastive loss. $\tau_1 = 1.0$ is the default choice of MoCov3 (Chen et al., 2021), and we do not modify it.

Another interesting phenomenon is that when the number of training epochs of SlimCLR-MoCov3 becomes larger, we need to increase the learning rate $(lr)$ when we start to train the sub-networks. The influence of learning rate for slimmable training in SlimCLR-MoCov3 is shown in Table 3b. Here the learning rate refers to the base learning rate, the immediate learning rate after the warm-up is calculated by this base learning rate and the current training steps: $\frac{1}{2}lr(1 + \cos(\pi \frac{step}{max_step}))$. Different from SlimCLR-MoCov2 and SlimCLR-MoCov3 with fewer epochs, SlimCLR-MoCov3 will get poor performance if we do not change the learning rate when training for more epochs. We attribute the difference to the LARS (You et al., 2017) optimizer we adopt for SlimCLR-MoCov3. LARS normalizes the gradient of layers in the networks to avoid the imbalance of gradient magnitude across layers and ensures convergence when training networks with very large batch size. LARS is sensitive to the change in learning rate and helps self-supervised models training with large batches converge fast (Chen et al., 2021; 2020a). When training for the 300 epochs, the full model can reach a local minima fast in the first 150 epochs. In this case, a learning rate 0.6 (half of the base learning rate) is not able to help the system walk out of the valley and reach a better local minima. Consequently, a large learning rate is needed to give the system more powerful momentum. From Table 3b, we can also see that SlimCLR-MoCov3 with LARS is sensitive to the change in learning rate. This is consistent with observations of previous works (Chen et al., 2021; 2020a).

## 5 Conclusion

In this work, we adapt slimmable networks for contrastive learning to obtain pre-trained small models in a self-supervised manner. By using slimmable networks, we can pre-train for one time and get several models with different sizes which are suitable for various devices with different computation resources. Besides, unlike previous distillation based methods, our methods do not require additional training process of large teacher models. However, weight-sharing sub-networks in a slimmable network cause severe interference to each other in self-supervised learning. One evidence of such interference we observed is the gradient imbalance in the backpropagation process. We develop several techniques to relieve the interference of weight-sharing networks during pre-training and linear evaluation. Two specific algorithms are instantiated in this work, *i.e.*, SlimCLR-MoCov2 and SlimCLR-MoCov3. We take extensive experiments on ImageNet and achieve better performance than previous arts with fewer network parameters and FLOPs.

## REFERENCES


Philip Bachman, R. Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *NeurIPS*, 2019.

Dana H. Ballard. Modular learning in neural networks. In *AAAI*, 1987.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *CVPR*, 2022.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. Big self-supervised models are strong semi-supervised learners. In *NeurIPS*, 2020b.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *CVPR*, 2021.

Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020c. URL https://arxiv.org/abs/2003.04297.

Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *ICCV*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *TPAMI*, 2016.

Zhiyuan Fang, Jianfeng Wang, Lijuan Wang, Lei Zhang, Yezhou Yang, and Zicheng Liu. SEED: self-supervised distillation for visual representation. In *ICLR*, 2021.

Yuting Gao, Jia-Xin Zhuang, Shaohui Lin, Hao Cheng, Xing Sun, Ke Li, and Chunhua Shen. Disco: Remedy self-supervised learning on lightweight models with distilled contrastive learning. In *ECCV*, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

Jia Guo. Reducing the teacher-student gap via adaptive temperatures, 2022. URL https://openreview.net/forum?id=h-z_zqT2yJU.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Soroush Abbasi Koohpayegani, Ajinkya Tejankar, and Hamed Pirsiavash. Compress: Self-supervised learning by compressing representations. In *NeurIPS*, 2020.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, 2021.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Ds-net++: Dynamic weight slicing for efficient inference in cnns and transformers. *T-PAMI*, 2022.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *CoRR*, abs/2006.08218, 2020. URL https://arxiv.org/abs/2006.08218.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR*, 2018.

Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Nenad Tomasev, Ioana Bica, Brian McWilliams, Lars Buesing, Razvan Pascanu, Charles Blundell, and Jovana Mitrovic. Pushing the limits of self-supervised resnets: Can we outperform supervised learning without labels on imagenet? *CoRR*, abs/2201.05119, 2022. URL https://arxiv.org/abs/2201.05119.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *NeurIPS*, 2016a.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016b.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *ICML*, 2021.

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, 2018.

Haohang Xu, Jiemin Fang, Xiaopeng Zhang, Lingxi Xie, Xinggang Wang, Wenrui Dai, Hongkai Xiong, and Qi Tian. Bag of instances aggregation boosts self-supervised distillation. In *ICLR*, 2022.

Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew R. Willis. Mutualnet: Adaptive convnet via mutual learning from network width and resolution. In *ECCV*, 2020.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

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

Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *ICCV*, 2019b.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *ICLR*, 2019.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *ECCV*, 2016.

Borui Zhao, Quan Cui, Renjie Song, Yiyu Qiu, and Jiajun Liang. Decoupled knowledge distillation. In *CVPR*, 2022a.

Shuai Zhao, Liguang Zhou, Wenxiao Wang, Deng Cai, Tin Lun Lam, and Yangsheng Xu. Towards better accuracy-efficiency trade-offs: Divide and co-training. *TIP*, 2022b.

# A APPENDIX

## A.1 CONDITIONS OF INPUTS GIVEN A SLIMMABLE LINEAR LAYER

We consider the conditions of inputs when only using one slimmable linear transformation layer, *i.e.*, consider solving multiple multi-class linear regression problems with shared weights. The parameters of the linear layer are $\boldsymbol{\theta} \in \mathbb{R}^{d \times C}$, $C$ is the number of classes, where $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_{11} \\ \boldsymbol{\theta}_{21} \end{bmatrix}$, $\boldsymbol{\theta}_{11} \in \mathbb{R}^{d_1 \times C}$, $\boldsymbol{\theta}_{21} \in \mathbb{R}^{d_2 \times C}$, $d_1 + d_2 = d$.

The first input for the full model is $\boldsymbol{X} \in \mathbb{R}^{N \times d}$, where $N$ is the number of samples, $\boldsymbol{X} = [\boldsymbol{X}_{11} \quad \boldsymbol{X}_{12}]$, $\boldsymbol{X}_{11} \in \mathbb{R}^{N \times d_1}$, $\boldsymbol{X}_{12} \in \mathbb{R}^{N \times d_2}$. The second input $\boldsymbol{X}_1 \in \mathbb{R}^{N \times d_1}$ is the input feature for the sub-model parameterized by $\boldsymbol{\theta}_{11}$.

Generally, we have $N \geq d > d_1$. We assume that both $\boldsymbol{X}$ and $\boldsymbol{X}_1$ have independent columns, *i.e.*, $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{X}_1^T\boldsymbol{X}_1$ are invertible. The ground truth is $\boldsymbol{T} \in \mathbb{R}^{N \times C}$.

The prediction of the full model is $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\theta}$, to minimize the sum-of-least-squares loss between prediction and ground truth, we get

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{T}\|_2^2. \tag{7}$$

By setting the derivative *w.r.t.* $\boldsymbol{\theta}$ to 0, we get

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{T}. \tag{8}$$

In the same way, we can get

$$\boldsymbol{\theta}_{11} = \left(\boldsymbol{X}_1^T\boldsymbol{X}_1\right)^{-1}\boldsymbol{X}_1^T\boldsymbol{T}. \tag{9}$$

For $\boldsymbol{X}^T\boldsymbol{X}$, we have

$$\boldsymbol{X}^T\boldsymbol{X} = [\boldsymbol{X}_{11} \quad \boldsymbol{X}_{12}]^T [\boldsymbol{X}_{11} \quad \boldsymbol{X}_{12}] = \begin{bmatrix} \boldsymbol{X}_{11}^T\boldsymbol{X}_{11} & \boldsymbol{X}_{11}^T\boldsymbol{X}_{12} \\ \boldsymbol{X}_{12}^T\boldsymbol{X}_{11} & \boldsymbol{X}_{12}^T\boldsymbol{X}_{12} \end{bmatrix}. \tag{10}$$

We denote the inverse of $\boldsymbol{X}^T\boldsymbol{X}$ is $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$, $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{B} = \boldsymbol{I}$, as $\boldsymbol{X}^T\boldsymbol{X}$ is a symmetric matrix, its inverse is also symmetric, so $\boldsymbol{B}_{12} = \boldsymbol{B}_{21}^T$. For $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{B} = \boldsymbol{I}$, we have

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{B} = \begin{bmatrix} \boldsymbol{X}_{11}^T\boldsymbol{X}_{11} & \boldsymbol{X}_{11}^T\boldsymbol{X}_{12} \\ \boldsymbol{X}_{12}^T\boldsymbol{X}_{11} & \boldsymbol{X}_{12}^T\boldsymbol{X}_{12} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{d_1} & \boldsymbol{0}_{d_1,\, d_2} \\ \boldsymbol{0}_{d_2,\, d_1} & \boldsymbol{I}_{d_2} \end{bmatrix}. \tag{11}$$

Then we can get

$$\boldsymbol{X}_{11}^T\boldsymbol{X}_{11}\boldsymbol{B}_{11} + \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\boldsymbol{B}_{21} = \boldsymbol{I}_{d_1}, \tag{12}$$

$$\boldsymbol{X}_{11}^T\boldsymbol{X}_{11}\boldsymbol{B}_{12} + \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\boldsymbol{B}_{22} = \boldsymbol{0}_{d_1,\, d_2}, \tag{13}$$

$$\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\boldsymbol{B}_{11} + \boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\boldsymbol{B}_{21} = \boldsymbol{0}_{d_1,\, d_2}, \tag{14}$$

$$\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\boldsymbol{B}_{12} + \boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\boldsymbol{B}_{22} = \boldsymbol{I}_{d_2}. \tag{15}$$

At the same time

$$\begin{aligned} \boldsymbol{\theta} &= \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{T} = \boldsymbol{B}\boldsymbol{X}^T\boldsymbol{T} \\ &= \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix} [\boldsymbol{X}_{11} \quad \boldsymbol{X}_{12}]^T \boldsymbol{T} \\ &= \begin{bmatrix} \boldsymbol{B}_{11}\boldsymbol{X}_{11}^T + \boldsymbol{B}_{12}\boldsymbol{X}_{12}^T \\ \boldsymbol{B}_{21}\boldsymbol{X}_{11}^T + \boldsymbol{B}_{22}\boldsymbol{X}_{12}^T \end{bmatrix} \boldsymbol{T}, \end{aligned} \tag{16}$$

and

$$\boldsymbol{\theta}_{11} = \left(\boldsymbol{B}_{11}\boldsymbol{X}_{11}^T + \boldsymbol{B}_{12}\boldsymbol{X}_{12}^T\right)\boldsymbol{T} = \left(\boldsymbol{X}_1^T\boldsymbol{X}_1\right)^{-1}\boldsymbol{X}_1^T\boldsymbol{T}. \tag{17}$$

From equation 14, we get

$$\boldsymbol{B}_{21} = -\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\boldsymbol{B}_{11}, \tag{18}$$

$$\boldsymbol{B}_{12} = -\boldsymbol{B}_{11}\boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}. \tag{19}$$

Substitute equation 18 into equation 12, we get

$$\boldsymbol{X}_{11}^T\boldsymbol{X}_{11}\boldsymbol{B}_{11} - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\boldsymbol{B}_{11} = \boldsymbol{I}_{d_1}, \tag{20}$$

$$\boldsymbol{B}_{11} = \left(\boldsymbol{X}_{11}^T\boldsymbol{X}_{11} - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\right)^{-1}. \tag{21}$$

At the same time

$$\begin{aligned}\boldsymbol{\theta}_{11} &= \left(\boldsymbol{B}_{11}\boldsymbol{X}_{11}^T + \boldsymbol{B}_{12}\boldsymbol{X}_{12}^T\right)\boldsymbol{T} \\ &= \boldsymbol{B}_{11}\left(\boldsymbol{X}_{11}^T - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\right)\boldsymbol{T} \\ &= \left(\boldsymbol{X}_{11}^T\boldsymbol{X}_{11} - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\right)^{-1}\left(\boldsymbol{X}_{11}^T - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\right)\boldsymbol{T}.\end{aligned} \tag{22}$$

Combining equation 17 and equation 22, we get the condition of the input

$$\left(\boldsymbol{X}_{11}^T\boldsymbol{X}_{11} - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\boldsymbol{X}_{11}\right)^{-1}\left(\boldsymbol{X}_{11}^T - \boldsymbol{X}_{11}^T\boldsymbol{X}_{12}\left(\boldsymbol{X}_{12}^T\boldsymbol{X}_{12}\right)^{-1}\boldsymbol{X}_{12}^T\right)\boldsymbol{T} = \left(\boldsymbol{X}_1^T\boldsymbol{X}_1\right)^{-1}\boldsymbol{X}_1^T\boldsymbol{T}. \tag{23}$$

### A.2 Influence of temperatures during distillation

In this section, we will analyze the influence of temperatures when applying distillation. One of the previous methods SEED (Fang et al., 2021) uses different temperatures for the student and teacher, without loss of generalization, we will also adopt a such strategy in our analysis. Specifically, we adopt $\tau_t$ for teacher and $\tau_s$ for student.

The predicted probability for a certain category $i$ of the student is $q_i = \frac{e^{z_i \,/\, \tau_s}}{\sum_j e^{z_j \,/\, \tau_s}}$, where $z$ is the output of the student model, *i.e.*, logit of the model. The probability for a certain category $i$ of the teacher is $p_i = \frac{e^{v_i \,/\, \tau_t}}{\sum_j e^{v_j \,/\, \tau_t}}$, where $v$ is the output of the student model. The loss is the KL divergence:

$$\mathcal{L} = -\sum_k p_k \log q_k. \tag{24}$$

The gradient of $q$ *w.r.t.* $z$ is:

$$\frac{\partial q_i}{\partial z_i} = \frac{\frac{1}{\tau_s}e^{z_i \,/\, \tau_s}\sum_j e^{z_j \,/\, \tau_s} - \frac{1}{\tau_s}e^{z_i \,/\, \tau_s}e^{z_i \,/\, \tau_s}}{\left(\sum_j e^{z_j \,/\, \tau_s}\right)^2} = \frac{1}{\tau_s}\left(q_i - q_i^2\right), \tag{25}$$

$$\frac{\partial q_t}{\partial z_i}|_{t\neq i} = \frac{0 - \frac{1}{\tau_s}e^{z_t \,/\, \tau_s}e^{z_i \,/\, \tau_s}}{\left(\sum_j e^{z_j \,/\, \tau_s}\right)^2} = -\frac{1}{\tau_s}q_i q_t. \tag{26}$$

Similarly,

$$\frac{\partial p_i}{\partial v_i} = \frac{1}{\tau_t}\left(p_i - p_i^2\right), \tag{27}$$

$$\frac{\partial p_t}{\partial v_i} = -\frac{1}{\tau_t}p_i p_t. \tag{28}$$

The gradient of $\mathcal{L}$ *w.r.t.* $z$ is:

$$\frac{\partial \mathcal{L}}{\partial z_i} = -\frac{p_i}{q_i}\frac{\partial q_i}{\partial z_i} + \sum_{t \neq i} -\frac{p_t}{q_t}\frac{\partial q_t}{\partial z_i} \tag{29}$$

$$= \frac{1}{\tau_s}(q_i - p_i) \tag{30}$$

$$= \frac{1}{\tau_s}\left(\frac{e^{z_i \,/\, \tau_s}}{\sum_j e^{z_j \,/\, \tau_s}} - \frac{e^{v_i \,/\, \tau_t}}{\sum_j e^{v_j \,/\, \tau_t}}\right). \tag{31}$$

Following classical KD (Hinton et al., 2015), we assume temperatures are much larger than logits and use the first-order Taylor series to approximate the exponential function:

$$\frac{\partial \mathcal{L}}{\partial z_i} \approx \frac{1}{\tau_s}\left(\frac{1 + \frac{z_i}{\tau_s}}{C + \sum_j \frac{z_j}{t_s}} - \frac{1 + \frac{v_i}{\tau_t}}{C + \sum_j \frac{v_j}{t_t}}\right), \tag{32}$$

where $C$ is the number of classes. Following classical KD (Hinton et al., 2015), we further assume $\sum_j z_j = \sum_j v_j = 0$, we can get:

$$\frac{\partial \mathcal{L}}{\partial z_i} \approx \frac{1}{C\tau_s}\left(\frac{z_i}{\tau_s} - \frac{v_i}{\tau_t}\right). \tag{33}$$

## A.3 Visualization

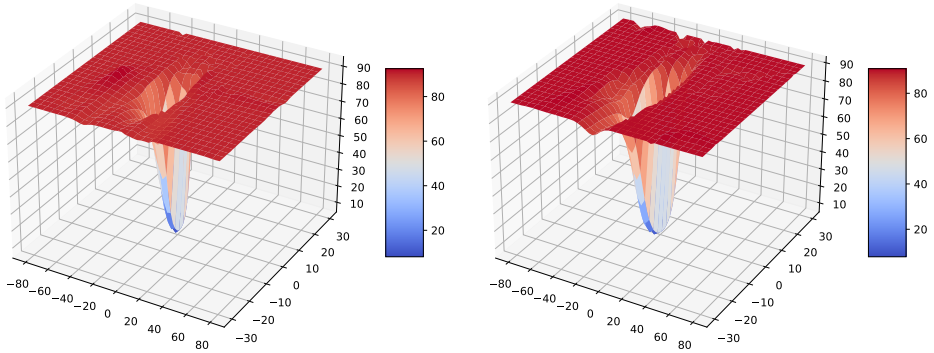
### A.3.1 Error surface and optimization trajectory

The performance degradation caused by the interference between weight-sharing networks is more severe in self-supervised learning compared to supervised learning. We already know that the gradient imbalance is more significant in self-supervised cases. Besides, the optimization process of self-supervised learning maybe also harder, so the gradient directions of weight-sharing networks may diverge from each other greatly and make the training process chaotic. In supervised cases, consistent global guidance relieves such divergence.

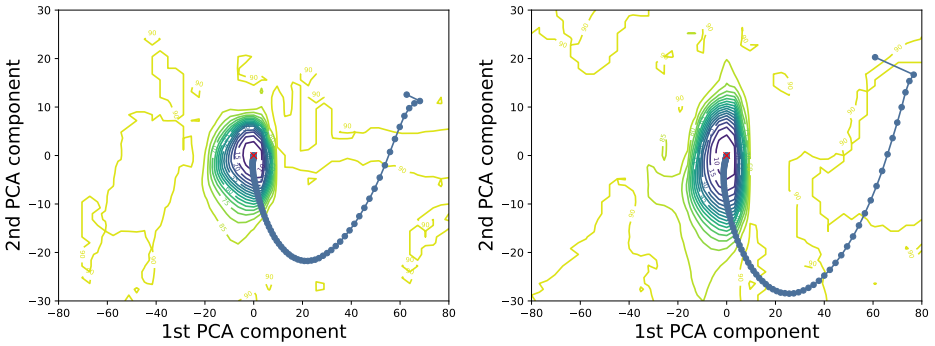
To better understand the interference between weight-sharing sub-networks in a slimmable network during optimization, we visualize the error surface and optimization trajectory using the method proposed by Li et al. (2018). We train slimmable networks in both supervised and self-supervised (MoCo (He et al., 2020)) manners on CIFAR-10 (Krizhevsky & Hinton, 2009). The base network is a ResNet-20×4, which has 4.3M parameters. We train the model for 100 epochs. At the end of each epoch, we save the weights of the full model and calculate the Top-1 accuracy. For self-supervised cases, we use a $k$-NN predictor (Wu et al., 2018) to obtain the accuracy. After training, we calculate the principle components of the differences of the saved weights at each epoch and the weights of the final model following Li et al. (2018). Then we use the first two principle components as the directions to plot the error surface and optimization trajectory in Figure 4.

The visualization shows that self-supervised learning is harder than supervised learning. In the left error surface in Figure 4a and Figure 4c, we can see that the terrain around the valley is flat in supervised cases; by contrast, the terrain around the valley is more complicated in self-supervised cases. From the trajectory of ResNet-20×4 in the left of Figure 4b and Figure 4d, the contours in supervised cases are denser, *i.e.*, the nearby two contours are closer. Namely, the model in self-supervised cases costs more time to achieve the same improvement of accuracy compared to the model in supervised cases (the gaps between two contour lines are all the same). In supervised cases, clear global guidance helps the model quickly reach the global minima. In self-supervised cases, it is harder for the model to reach the global minima fast without such global guidance.

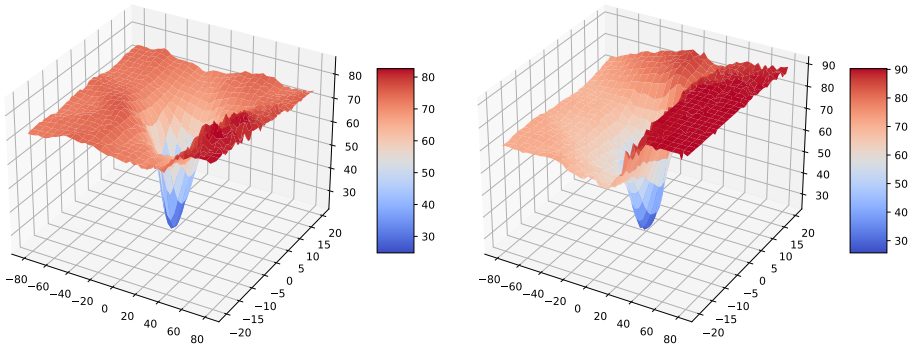
The visualization shows that the interference of weight-sharing networks is more significant in self-supervised cases. First of all, in self-supervised cases, weight-sharing networks bring huge changes to the error surface in Figure 4c. In contrast, the change is not so obvious in supervised cases. Second, the interference between weight-sharing networks in self-supervised cases makes the model shift more away from the global minima (the origin in the visualization) as shown in Figure 4d. In Figure 4b, the maximal offsets from the global minima along the 2nd PCA component are 21.75 and



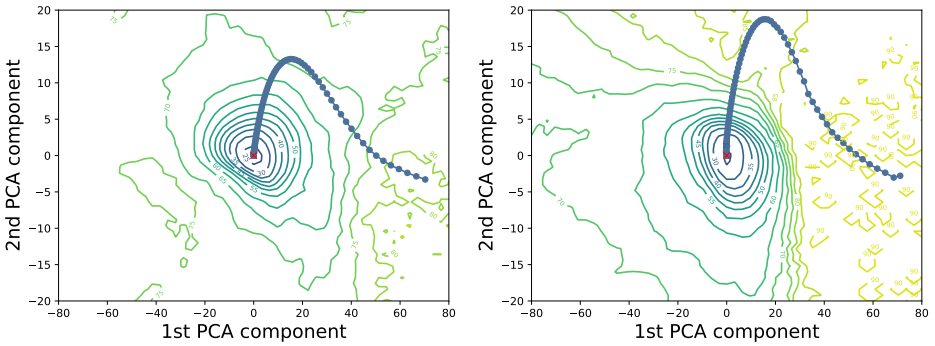
(a) error surface (supervised). left: R-20×4, 94.07%; right: R-20×4$_{[1.0,0.5]}$, 93.58%, 93.03%.




(b) trajectory (supervised). left: R-20×4, 94.07%; right: R-20×4$_{[1.0,0.5]}$, 93.58%, 93.03%.

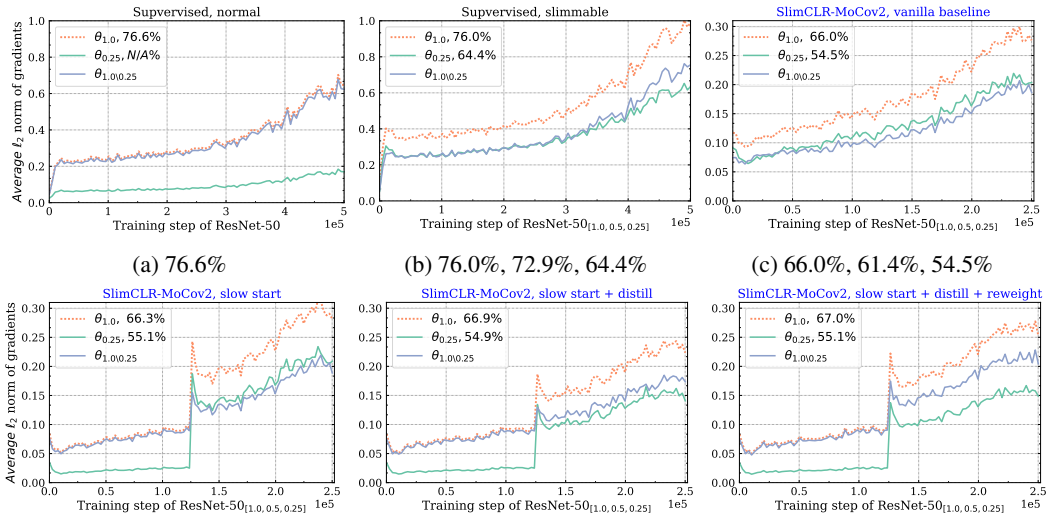


(c) error surface (MoCo). left: R-20×4, 76.75%; right: R-20×4$_{[1.0,0.5]}$, 75.74%, 74.24%.




(d) trajectory (MoCo). left: R-20×4, 76.75%; right: R-20×4$_{[1.0,0.5]}$, 75.74%, 74.24%.

Figure 4: Visualization of the error surface and optimization trajectory.




(a) 76.6% (b) 76.0%, 72.9%, 64.4% (c) 66.0%, 61.4%, 54.5%

(d) 66.3%, 62.2%, 55.1% (e) 66.9%, 62.3%, 54.9% (f) 67.0%, 62.4%, 55.1%

Figure 5: The average gradient norm. We calculate the $\ell_2$ norm of gradients layer-wise and get their mean. $\theta_{1.0} \in \Theta_{1.0}$ and $\theta_{0.25} \in \Theta_{0.25}$ represent parameters of the full model and sub-network with width 0.25. $\theta_{1.0\backslash 0.25} \in \Theta_{1.0} \backslash \Theta_{0.25}$, denote the rest parameters of $\Theta_{1.0}$ besides $\Theta_{0.25}$. It is normal that the gradient norm increases during training (Goodfellow et al., 2016).

28.49 for ResNet-20×4 and ResNet-20×4$_{[1.0,0.5]}$. The offset increased 31.0%. For self-supervised cases in Figure 4d, the maximal offsets from the global minima along the 2nd PCA component are 13.26 and 18.75 for ResNet-20×4 and ResNet-20×4$_{[1.0,0.5]}$. The offset increased 41.4%. It is clear that the interference of weight-sharing networks is more significant in self-supervised cases compared to supervised cases.

### A.3.2 Gradient imbalance

Besides the ratios of gradient norms of a slimmable network in Figure 3, we also display the value of gradient norms in Figure 5 to help us better understand the gradient imbalance.

## A.4 More implementation details

**Slimmable networks** We adopt the implementation of slimmable networks described in Yu et al. (2019), which has switchable batch normalization layers. Namely, each network in the slimmable network has its own independent batch normalization process.

**SlimCLR-MoCov2** We train SlimCLR-MoCov2 on 8 Tesla V100 32GB GPUs without synchronized batch normalization across GPUs. The momentum coefficient $m$ is 0.999 during training.

**SlimCLR-MoCov3** We train SlimCLR-MoCov3 on 8 Tesla V100 32GB GPUs with synchronized batch normalization across GPUs. Synchronized batch normalization is important for MoCov3 to obtain a better performance in linear evaluation. The momentum coefficient $m$ is 0.99 with a cosine schedule when training for 300 epochs. The data augmentations are the same as augmentations of MoCov3 Chen et al. (2021).