Robust and reliable large-scale transfer learning

Mitchell Wortsman

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2023

Reading Committee:
Ali Farhadi, Co-Chair
Ludwig Schmidt, Co-Chair
Luke Zettlemoyer

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Robust and reliable large-scale transfer learning

Mitchell Wortsman

Co-chairs of the Supervisory Committee:

Professor Ali Farhadi
Computer Science and Engineering

Assistant Professor Ludwig Schmidt
Computer Science and Engineering

Machine learning is currently witnessing a convergence towards large, pre-trained models that are fine-tuned for specific applications such as chat. This process, known as large-scale transfer learning, increasingly produces models that are deployed in real-world applications. It is therefore imperative that large-scale transfer is robust and reliable. Our research towards this goal advances fine-tuning robustness and pre-training reliability. Towards robust fine-tuning, we establish weight-interpolation as a technique to combine specialist models into one general model. We use this method to address the tension between robustness and accuracy that can emerge when fine-tuning. Next, we extend this technique to multiple models fine-tuned with diverse hyperparameters to obtain a new state-of-the-art on ImageNet. Towards reliable pre-training, we address a key obstacle that emerges at large scale—training instability. We uncover a predictive relationship between large updates in the network's first layer and loss spikes which slow or destabilize learning. Finally, we establish small-scale proxy models as a reliable tool for studying training divergence, allowing us to predict and mitigate instabilities before they emerge. Our results indicate multiple promising directions for future development, from decentralized training to improvements in model architecture.

Acknowledgements

Thank you to my advisors, mentors, friends, and family for their support, and for teaching me everything.

DEDICATION

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today's best-performing neural networks result from a two-step procedure: first, *pre-train* a model on a large heterogeneous dataset, next *fine-tune* to further adapt the model to a task of interest. This is also referred to as transfer learning [97, 285, 142, 139]. While transfer learning is not new, it has continued to deliver remarkable success when applied at successively larger scales [210, 33, 202, 195].

While scale has driven progress in transfer learning, it has also uncovered a new and unique set of challenges. For one, researchers have reported *training instabilities* at large scale that did not occur when using the same hyperparameters at small-scale [44, 58]. For another, while large pre-trained models perform well on a broad range of inputs, fine-tuning can deteriorate this robustness.

The goal of this dissertation is to advance large-scale transfer learning as a robust and reliable paradigm in machine learning. Towards this goal we present better algorithms for fine-tuning which improve robustness by interpolating weights (Chapters 2–4). Moreover, we promote pre-training reliability via a mechanistic understanding of the phenomena which underlie training instabilities (Chapters 6–7).

Our exploration in both aforementioned directions also suggests promising areas for improving models. For one, the success of weight-interpolation indicates promise for decentralized learning (Chapter 5), which is essential at scales where accelerators cannot all be co-located. Moreover, our research on understanding instability uncovers a neural network architecture that matches self-attention [257] while presenting new opportunities for parallelism (Chapter 8).

Robust fine-tuning. Our work in this direction began with an open problem raised by several researchers [210, 202, 6]: while fine-tuning increases accuracy on the target distribution, it also reduces performance under distribution shift. This naturally led to the following question: *can we design fine-tuning methods which preserve the robustness of the pre-trained model?*

Our solution to this problem leverages the observation that fine-tuned models often appear to lie in a single low error region [192]. To improve robustness, we therefore interpolate the weights of the pre-trained and fine-tuned models [272] (Chapter 2). This achieves the best of both worlds: capturing the robustness of the pre-trained model and the in-distribution accuracy of the fine-tuned model. Surprisingly, network weights can be linearly combined despite nonlinear activation functions.

Next, we introduce model soups [273] which generalizes the aforementioned method to interpolate the weights of *multiple* fine-tuned models (Chapter 3). The conventional procedure for maximizing model performance is to try many different hyperparameters, then select the best model and discard the remainder. Model soups proposes an alternative to this procedure in the context of fine-tuning: we average the weights of multiple fine-tuned models, and are often able to produce a better model with no added inference cost. Model soups achieved state-of-the-art on ImageNet [59] and two datasets from WILDS [138]. Finally, Chapters 4 and 5 apply weight-interpolation for model patching and decentralized training, respectively.

Reliable large-scale pre-training. Scaling up Transformer pre-training has resulted in substantial advancements. However, when training large Transformers, researchers have reported instabilities which slow or destabilize learning [44, 58, 301, 187, 49]. For scale to continue to produce progress, understanding and mitigating transformer training pathologies is critical.

Our work in this direction addresses two types of instabilities, *fast* loss-spikes (Chapter 6) and *slow* training divergences (Chapter 7). In both cases, we uncover a mechanistic explanation underlying certain failures.

First, we establish a predictive relationship between large updates in the neural network's first layer and *fast* loss-spikes when using the standard Adam optimizer [136]. Large updates occur because the second moment EMA is small compared to the current gradient, e.g., when processing rare token embeddings. This motivates an Adam-AdaFactor [234] hybrid, which we develop and test in Chapter 6. In addition, Chapter 6 advances eight-bit training of large neural networks, a crucial step towards efficiency as pre-training becomes

prohibitively expensive.

Next, we establish small-scale proxy models as a reliable tool for studying *slow* training divergences at large-scale (Chapter 7). As a highlight, we demonstrate that small-scale experiments can predict when and how a large-scale instability will occur. Moreover, we use small-scale experiments to uncover a new instabilitiy that has not previously been reported—vanishing updates due to the Adam ϵ hyperparamter exceeding the second moment EMA.

Other directions. The work we've discussed so far is made possible by our research in related directions. Concretely, our work on neural network loss landscapes [216, 269, 271] is foundational for the weight-interpolation approaches we've developed for robust fine-tuning. Moreover, our efforts democratizing large-scale pre-training via Open{CLIP, Flamingo, LM} [120, 9, 105] and next-generation large-scale datasets [230, 87] motivated and facilitated our research on pre-training.

Chapter 2

Robust fine-tuning of zero-shot models

2.1 Overview

Large pre-trained models such as CLIP or ALIGN offer consistent accuracy across a range of data distributions when performing zero-shot inference (i.e., without fine-tuning on a specific dataset). Although existing fine-tuning methods substantially improve accuracy on a given target distribution, they often reduce robustness to distribution shifts. We address this tension by introducing a simple and effective method for improving robustness while fine-tuning: ensembling the weights of the zero-shot and fine-tuned models (WiSE-FT). Compared to standard fine-tuning, WiSE-FT provides large accuracy improvements under distribution shift, while preserving high accuracy on the target distribution. On ImageNet and five derived distribution shifts, WiSE-FT improves accuracy under distribution shift by 4 to 6 percentage points (pp) over prior work while increasing ImageNet accuracy by 1.6 pp. WiSE-FT achieves similarly large robustness gains (2 to 23 pp) on a diverse set of six further distribution shifts, and accuracy gains of 0.8 to 3.3 pp compared to standard fine-tuning on seven commonly used transfer learning datasets. These improvements come at no additional computational cost during fine-tuning or inference.

2.2 Introduction

A foundational goal of machine learning is to develop models that work reliably across a broad range of data distributions. Over the past few years, researchers have proposed a variety of distribution shifts on which

current algorithmic approaches to enhance robustness yield little to no gains [250, 186]. While these negative results highlight the difficulty of learning robust models, large pre-trained models such as CLIP [210], ALIGN [125] and BASIC [202] have recently demonstrated unprecedented robustness to these challenging distribution shifts. The success of these models points towards pre-training on large, heterogeneous datasets as a promising direction for increasing robustness. However, an important caveat is that these robustness improvements are largest in the zero-shot setting, i.e., when the model performs inference without fine-tuning on a specific target distribution.

In a concrete application, a zero-shot model can be fine-tuned on extra application-specific data, which often yields large performance gains on the target distribution. However, in the experiments of Radford et al. [210] and Pham et al. [202], fine-tuning comes at the cost of robustness: across several natural distribution shifts, the accuracy of their fine-tuned models is lower than that of the original zero-shot model. This leads to a natural question:

Can zero-shot models be fine-tuned without reducing accuracy under distribution shift?

As pre-trained models are becoming a cornerstone of machine learning, techniques for fine-tuning them on downstream applications are increasingly important. Indeed, the question of robustly fine-tuning pre-trained models has recently also been raised as an open problem by several authors [6, 26, 210, 202]. Andreassen et al. [6] explored several fine-tuning approaches but found that none yielded models with improved robustness at high accuracy. Furthermore, Taori et al. [250] demonstrated that no current algorithmic robustness interventions provide consistent gains across the distribution shifts where zero-shot models excel.

In this paper, we conduct an empirical investigation to understand and improve fine-tuning of zero-shot models from a distributional robustness perspective. We begin by measuring how different fine-tuning approaches (last-layer vs. end-to-end fine-tuning, hyperparameter changes, etc.) affect the accuracy under distribution shift of the resulting fine-tuned models. Our empirical analysis uncovers two key issues in the standard fine-tuning process. First, the robustness of fine-tuned models varies substantially under even small changes in hyperparameters, but the best hyperparameters cannot be inferred from accuracy on the target distribution alone. Second, more aggressive fine-tuning (e.g., using a larger learning rate) yields larger accuracy improvements on the target distribution, but can also reduce accuracy under distribution shift by a large amount.

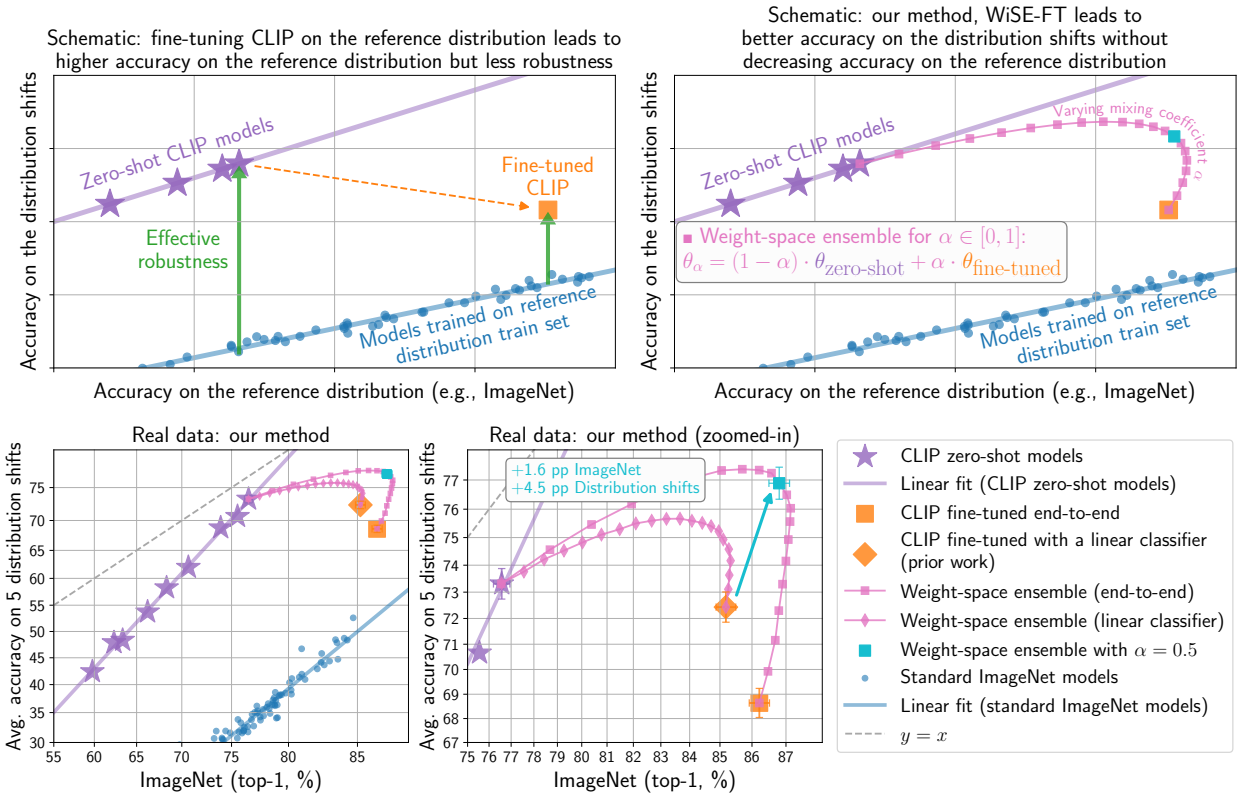

Figure 2.1: (Top left) Zero-shot CLIP models exhibit moderate accuracy on the reference distribution (x-axis, the target for fine-tuning) and high effective robustness (accuracy on the distribution shifts beyond the baseline models). In contrast, standard fine-tuning—either end-to-end or with a linear classifier (final layer)—attains higher accuracy on the reference distribution but less effective robustness. **(Top right)** Our method linearly interpolates between the zero-shot and fine-tuned models with a mixing coefficient $\alpha \in [0, 1]$. **(Bottom)** On five distribution shifts derived from ImageNet (ImageNetV2, ImageNet-R, ImageNet Sketch, ObjectNet, and ImageNet-A), WiSE-FT improves average accuracy relative to both the zero-shot and fine-tuned models while maintaining or improving accuracy on ImageNet.

Motivated by the above concerns, we propose a robust way of fine-tuning zero-shot models that addresses the aforementioned trade-off and achieves the best of both worlds: increased performance under distribution shift while maintaining or even improving accuracy on the target distribution relative to standard fine-tuning. In addition, our method simplifies the choice of hyperparameters in the fine-tuning process.

Our method (Figure 2.1) has two steps: first, we fine-tune the zero-shot model on the target distribution. Second, we combine the original zero-shot and fine-tuned models by linearly interpolating between their weights, which we refer to as weight-space ensembling. Interpolating model parameters is a classical idea in convex optimization dating back decades (e.g., see [227, 205]). Here, we empirically study model interpolation for non-convex models from the perspective of distributional robustness. Interestingly, linear interpolation in weight-space still succeeds despite the non-linearity in the activation functions of the neural networks.

Weight-space ensembles for fine-tuning (WiSE-FT) substantially improve accuracy under distribution shift compared to prior work while maintaining high performance on the target distribution. Concretely, on ImageNet [59] and five of the natural distribution shifts studied by Radford et al. [210], WiSE-FT applied to standard end-to-end fine-tuning improves accuracy under distribution shift by 4 to 6 percentage points (pp) over prior work while maintaining or improving the ImageNet accuracy of the fine-tuned CLIP model. Relative to the zero-shot model, WiSE-FT improves accuracy under distribution shift by 1 to 9 pp. Moreover, WiSE-FT improves over a range of alternative approaches such as regularization and evaluating at various points throughout fine-tuning. These robustness gains come at no additional computational cost during fine-tuning or inference.

While our investigation centers around CLIP, we observe similar trends for other zero-shot models including ALIGN [125], BASIC [202], and a ViT model pre-trained on JFT [71]. For instance, WiSE-FT improves the ImageNet accuracy of a fine-tuned BASIC-L model by 0.4 pp, while improving average accuracy under distribution shift by 2 to 11 pp.

To understand the robustness gains of WiSE-FT, we first study WiSE-FT when fine-tuning a linear classifier (last layer) as it is more amenable to analysis. In this linear case, our procedure is equivalent to ensembling the outputs of two models, and experiments point towards the complementarity of model predictions as a key property. For end-to-end fine-tuning, we connect our observations to earlier work on

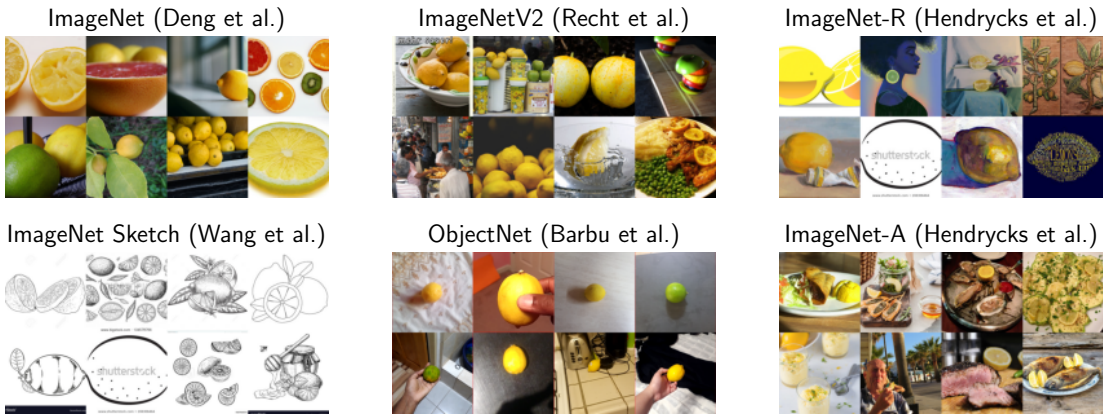

Figure 2.2: Samples of the class *lemon*, from the reference distribution ImageNet [59] and the derived distribution shifts considered in our main experiments: ImageNet-V2 [222], ImageNet-R [112], ImageNet Sketch [260], ObjectNet [17], and ImageNet-A [113].

the phenomenology of deep learning. Neyshabur et al. [192] found that end-to-end fine-tuning the same model twice yielded two different solutions that were connected via a linear path in weight-space along which error remains low, known as linear mode connectivity [82, 190]. Our observations suggest a similar phenomenon along the path generated by WiSE-FT, but the exact shape of the loss landscape and connection between error on the target and shifted distributions are still open problems.

In addition to the aforementioned ImageNet distribution shifts, WiSE-FT consistently improves robustness on a diverse set of six additional distribution shifts including: (i) geographic shifts in satellite imagery and wildlife recognition (WILDS-FMoW, WILDS-iWildCam) [138, 45, 19], (ii) reproductions of the popular image classification dataset CIFAR-10 with a distribution shift (CIFAR-10.1 and CIFAR-10.2) [222, 171], and (iii) datasets with distribution shift induced by temporal perturbations in videos (ImageNet-Vid-Robust and YTBB-Robust) [231]. Beyond the robustness perspective, WiSE-FT also improves accuracy compared to standard fine-tuning, reducing the relative error rate by 4-49% on a range of seven datasets: ImageNet, CIFAR-10, CIFAR-100 [144], Describable Textures [46], Food-101 [29], SUN397 [276], and Stanford Cars [143]. Even when fine-tuning data is scarce, reflecting many application scenarios, we find that WiSE-FT improves performance.

Overall, WiSE-FT is simple, universally applicable in the problems we studied, and can be implemented in a few lines of code. Hence we encourage its adoption for fine-tuning zero-shot models.

2.3 Background and experimental setup

Our experiments compare the performance of zero-shot models, corresponding fine-tuned models, and models produced by WiSE-FT. To measure robustness, we contrast model accuracy on two related but different distributions, a reference distribution $\mathcal{D}_{\text{ref}}$ which is the target for fine-tuning, and shifted distribution $\mathcal{D}_{\text{shift}}$.[1] We assume both distributions have test sets for evaluation, and $\mathcal{D}_{\text{ref}}$ has an associated training set $\mathcal{S}^{\text{tr}}_{\text{ref}}$ which is typically used for training or fine-tuning. The goal for a model is to achieve both high accuracy and consistent performance on the two distributions $\mathcal{D}_{\text{ref}}$ and $\mathcal{D}_{\text{shift}}$. This is a natural goal as humans often achieve similar accuracy across the distribution shifts in our study [232].

For a model f, we let $\mathsf{Acc}_{\text{ref}}(f)$ and $\mathsf{Acc}_{\text{shift}}(f)$ refer to classification accuracy on the reference and shifted test sets, respectively. We consider k-way image classification, where x_i is an image with corresponding label $y_i \in \{1, ..., k\}$. The outputs of f are k-dimensional vectors of non-normalized class scores.

Distribution shifts. Taori et al. [250] categorized distribution shifts into two broad categories: (i) *synthetic*, e.g., ℓ_∞-adversarial examples or artificial changes in image contrast, brightness, etc. [110, 25, 24, 92, 5]; and (ii) *natural*, where samples are not perturbed after acquisition and changes in data distributions arise through naturally occurring variations in lighting, geographic location, crowdsourcing process, image styles, etc. [250, 222, 112, 113, 138]. Following Radford et al. [209], our focus here is on natural distribution shifts as they are more representative of the real world when no active adversary is present. Specifically, we present our key results for five natural distribution shifts derived from ImageNet (i.e., $\mathcal{S}^{\text{tr}}_{\text{ref}}$ is ImageNet):

- ImageNet-V2 (IN-V2) [222], a reproduction of the ImageNet test set with distribution shift
- ImageNet-R (IN-R) [112], renditions (e.g., sculptures, paintings) for 200 ImageNet classes
- ImageNet Sketch (IN-Sketch) [260], which contains sketches instead of natural images
- ObjectNet [17], a test set of objects in various scenes with 113 classes overlapping with ImageNet
- ImageNet-A (IN-A) [113], a test set of natural images misclassified by a ResNet-50 [107] for 200 ImageNet classes.

[1] $\mathcal{D}_{\text{ref}}$ and $\mathcal{D}_{\text{shift}}$ are sometimes referred to as *in-distribution* (ID) and *out-of-distribution* (OOD). In this work, we include evaluations of zero-shot models, which are *not* trained on data from the reference distribution, so referring to $\mathcal{D}_{\text{ref}}$ would be imprecise. For clarity, we avoid the ID/OOD terminology.

Figure 2.2 illustrates the five distribution shifts.

Effective robustness and scatter plots. To compare the robustness of models with different accuracies on the reference distribution, we follow the *effective robustness* framework introduced by Taori et al. [250]. Effective robustness quantifies robustness as accuracy *beyond a baseline* trained only on the reference distribution. A useful tool for studying (effective) robustness are scatter plots that illustrate model performance under distribution shift [222, 250]. These scatter plots display accuracy on the reference distribution on the x-axis and accuracy under distribution shift on the y-axis, i.e., a model f is shown as a point $(\mathsf{Acc}_{\text{ref}}(f),\ \mathsf{Acc}_{\text{shift}}(f))$. Figure 2.1 exemplifies these scatter plots with both schematics and real data. For the distribution shifts we study, accuracy on the reference distribution is a reliable predictor of accuracy under distribution shift [250, 186]. In other words, there exists a function $\beta : [0,1] \to [0,1]$ such that $\mathsf{Acc}_{\text{shift}}(f)$ approximately equals $\beta(\mathsf{Acc}_{\text{ref}}(f))$ for models f trained on the train set $\mathcal{S}^{\text{tr}}_{\text{ref}}$. Effective robustness [250] is accuracy beyond this baseline, defined formally as $\rho(f) = \mathsf{Acc}_{\text{shift}}(f) - \beta(\mathsf{Acc}_{\text{ref}}(f))$.

In the corresponding scatter plots, effective robustness is vertical movement above expected accuracy under distribution shift (Figure 2.1, top). Effective robustness thereby disentangles accuracy changes on the reference distribution from the effect of robustness interventions. When we say that a model is robust to distribution shift, we mean that effective robustness is positive. Taori et al. [250] observed that no algorithmic robustness intervention consistently achieves substantial effective robustness across the distribution shifts in Figure 2.2—the first method to do so was zero-shot CLIP. Empirically, when applying logit (or probit) axis scaling, models trained on the reference distribution approximately lie on a linear trend [250, 186]. As in Taori et al. [250], we apply logit axis scaling and show 95% Clopper-Pearson confidence intervals for the accuracies of select points.

Zero-shot models and CLIP. We primarily explore CLIP models [210], although we also investigate other zero-shot models including ALIGN [125], BASIC [202] and a ViT model pre-trained on JFT [71]. Zero-shot models exhibit effective robustness and lie on a qualitatively different linear trend (Figure 2.1). CLIP-like models are pre-trained using image-caption pairs from the web. Given a set of image-caption pairs $\{(x_1, s_1)..., (x_B, s_B)\}$, CLIP-like models train an image-encoder g and text-encoder h such that the similarity $\langle g(x_i), h(s_i)\rangle$ is maximized relative to unaligned pairs. CLIP-like models perform zero-shot k-

way classification given an image x and class names $C = \{c_1, ..., c_k\}$ by matching x with potential captions. For instance, using caption $s_i =$ "a photo of a $\{c_i\}$" for each class i, the zero-shot model predicts the class via $\text{argmax}_j \langle g(x), h(s_j) \rangle$.[2] Equivalently, one can construct $\mathbf{W}_{\text{zero-shot}} \in \mathbb{R}^{d \times k}$ with columns $h(s_j)$ and compute outputs $f(x) = g(x)^\top \mathbf{W}_{\text{zero-shot}}$. Unless explicitly mentioned, our experiments use the CLIP model `ViT-L/14@336px`, although all CLIP models are displayed in our scatter plots.

2.4 Weight-space ensembles for fine-tuning

This section describes and motivates our proposed method, WiSE-FT, which consists of two simple steps. First, we fine-tune the zero-shot model on application-specific data. Second, we combine the original zero-shot and fine-tuned models by linearly interpolating between their weights, also referred to as weight-space ensembling. WiSE-FT can be implemented in a few lines of PyTorch.

The zero-shot model excels under distribution shift while standard fine-tuning achieves high accuracy on the reference distribution. Our motivation is to combine these two models into one that achieves the best of both worlds. Weight-space ensembles are a natural choice as they ensemble without extra computational cost. Moreover, previous work has suggested that interpolation in weight space may improve performance when models share part of their optimization trajectory [123, 192].

Step 1: Standard fine-tuning. As in Section 2.3, we let $\mathcal{S}^{\text{tr}}_{\text{ref}}$ denote the dataset used for fine-tuning and g denote the image encoder used by CLIP. We are now explicit in writing $g(x, \mathbf{V}_{\text{enc}})$ where x is an input image and $\mathbf{V}_{\text{enc}}$ are the parameters of the encoder g. Standard fine-tuning considers the model $f(x, \theta) = g\left(x, \mathbf{V}_{\text{enc}}\right)^\top \mathbf{W}_{\text{classifier}}$ where $\mathbf{W}_{\text{classifier}} \in \mathbb{R}^{d \times k}$ is the classification head and $\theta = [\mathbf{V}_{\text{enc}}, \mathbf{W}_{\text{classifier}}]$ are the parameters of f. We then solve $\text{argmin}_\theta \left\{ \sum_{(x_i, y_i) \in \mathcal{S}^{\text{tr}}_{\text{ref}}} \ell(f(x_i, \theta),\ y_i) + \lambda R(\theta) \right\}$ where ℓ is the cross-entropy loss and R is a regularization term (e.g., weight decay). We consider the two most common variants of fine-tuning: end-to-end, where all values of θ are modified, and fine-tuning only a linear classifier, where $\mathbf{V}_{\text{enc}}$ is fixed at the value learned during pre-training.

[2] For improved accuracy, the embedding of a few candidate captions are averaged, e.g., $s_i^{(1)}$="a *photo* of a $\{c_i\}$" and $s_i^{(2)}$="a *picture* of a $\{c_i\}$" (referred to as prompt ensembling [210]).

Step 2: Weight-space ensembling. For a mixing coefficient $\alpha \in [0, 1]$, we consider the *weight-space ensemble* between the zero-shot model with parameters θ_0 and the model obtained via standard fine-tuning with parameters θ_1. The predictions of the weight-space ensemble wse are given by

$$\mathsf{wse}(x, \alpha) = f(x, (1 - \alpha) \cdot \theta_0 + \alpha \cdot \theta_1) \,, \tag{2.1}$$

i.e., we use the element-wise weighted average of the zero-shot and fined-tuned parameters. When fine-tuning only the linear classifier, weight-space ensembling is equivalent to the traditional output-space ensemble [66, 31, 84] $(1-\alpha)\cdot f(x, \theta_0)+\alpha\cdot f(x, \theta_1)$ since Equation 2.1 decomposes as $(1-\alpha)\cdot g(x, \mathbf{V}_{\text{enc}})^\top \mathbf{W}_{\text{zero-shot}}+ \alpha \cdot g(x, \mathbf{V}_{\text{enc}})^\top \mathbf{W}_{\text{classifier}}$.

As neural networks are non-linear with respect to their parameters, ensembling all layers—as we do when end-to-end fine-tuning—typically fails, achieving no better accuracy than a randomly initialized neural network [82]. However, as similarly observed by previous work where part of the optimization trajectory is shared [123, 82, 192], we find that the zero-shot and fine-tuned models are connected by a linear path in weight-space along which accuracy remains high (explored further in Section 2.6.2).

Remarkably, as we show in Section 2.5, WiSE-FT improves accuracy under distribution shift while maintaining high performance on the reference distribution relative to fine-tuned models. These improvements come without any additional computational cost as a single set of weights is used.

2.5 Results

This section presents our key experimental findings. First, we show that WiSE-FT boosts the accuracy of a fine-tuned CLIP model on five ImageNet distribution shifts studied by Radford et al. [210], while maintaining or improving ImageNet accuracy. Next, we present additional experiments, including more distribution shifts, the effect of hyperparameters, accuracy improvements on the reference distribution, and experiments in the low-data regime. Finally, we demonstrate that our findings are more broadly applicable by exploring WiSE-FT for BASIC [202], ALIGN [125], and a ViT-H/14 [71] model pre-trained on JFT-300M [245].

	IN (reference)	Distribution shifts					Avg shifts	Avg ref., shifts
		IN-V2	IN-R	IN-Sketch	ObjectNet*	IN-A		
CLIP `ViT-L/14@336px`								
Zero-shot [210]	76.2	70.1	88.9	60.2	70.0	77.2	73.3	74.8
Fine-tuned LC [210]	85.4	75.9	84.2	57.4	66.2	75.3	71.8	78.6
Zero-shot (PyTorch)	76.6	70.5	89.0	60.9	69.1	77.7	73.4	75.0
Fine-tuned LC (ours)	85.2	75.8	85.3	58.7	67.2	76.1	72.6	78.9
Fine-tuned E2E (ours)	86.2	76.8	79.8	57.9	63.3	65.4	68.6	77.4
WiSE-FT (ours)								
LC, α=0.5	83.7	76.3	89.6	63.0	70.7	79.7	75.9	79.8
LC, optimal α	85.3	76.9	89.8	63.0	70.7	79.7	75.9	80.2
E2E, α=0.5	86.8	**79.5**	89.4	64.7	71.1	79.9	76.9	81.8
E2E, optimal α	**87.1**	**79.5**	**90.3**	**65.0**	**72.1**	**81.0**	**77.4**	**81.9**

Table 2.1: Accuracy of various methods on ImageNet and derived distribution shifts for CLIP `ViT-L/14@336px` [210]. E2E: end-to-end; LC: linear classifier. *Avg shifts* displays the mean performance among the five distribution shifts, while *Avg reference, shifts* shows the average of ImageNet (reference) and Avg shifts. For optimal α, we choose the single mixing coefficient that maximizes the column.

Main results: ImageNet and associated distribution shifts. As illustrated in Figure 2.1, when the mixing coefficient α varies from 0 to 1, $\mathsf{wse}(\cdot, \alpha)$ is able to simultaneously improve accuracy on both the reference and shifted distributions. Table 2.1 presents our main results on ImageNet and five derived distribution shifts. WiSE-FT (end-to-end, α=0.5) outperforms numerous strong models in both average accuracy under distribution shift and the average accuracy on the reference and shifted distributions. While future work may lead to more sophisticated strategies for choosing the mixing coefficient α, α=0.5 yields close to optimal performance across a range of experiments. Hence, we recommend α=0.5 when no domain knowledge is available.

Robustness on additional distribution shifts. Beyond the five distribution shifts derived from ImageNet, WiSE-FT consistently improves robustness on a diverse set of further distributions shifts including geographic shifts in satellite imagery and wildlife recognition (WILDS-FMoW [138, 45], WILDS-iWildCam [138, 19]), reproductions of the popular image classification dataset CIFAR-10 [144] with a distribution shift (CIFAR-10.1 [222] and CIFAR-10.2 [171]), and datasets with distribution shift induced by temporal perturbations in videos (ImageNet-Vid-Robust and YTBB-Robust [232]). Concretely, WiSE-FT (α=0.5) improves performance under distribution shift by 3.5, 6.2, 1.7, 2.1, 9.0 and 23.2 pp relative to the fine-

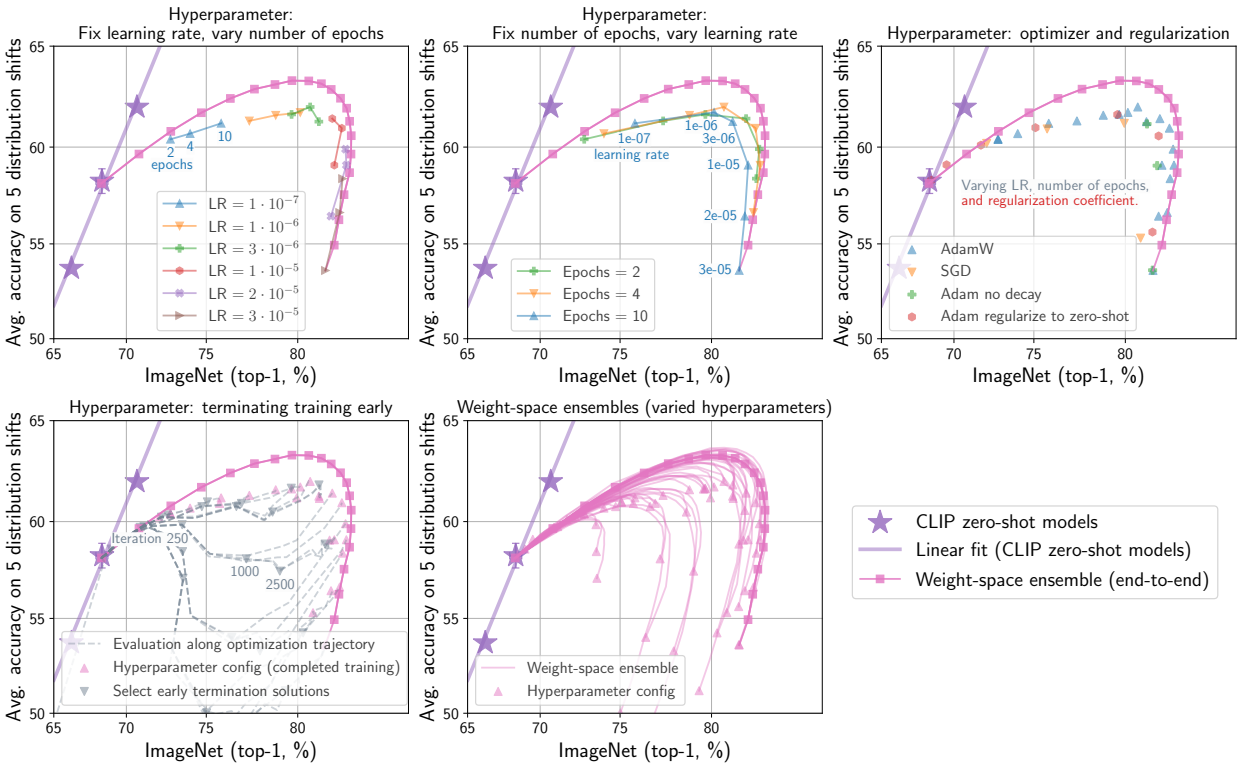

Figure 2.3: The robustness of fine-tuned models varies substantially under even small changes in hyperparameters. Applying WiSE-FT addresses this brittleness and can remove the trade-off between accuracy on the reference and shifted distributions. Results shown for CLIP `ViT-B/16` fine-tuned with cosine-annealing learning rate schedule and all models in the top left and top middle plots are fine-tuned with AdamW [169]. Moreover, *regularize to zero-shot* appends the regularizer $\lambda\|\theta - \theta_0\|_2^2$ to the fine-tuning objective, where θ_0 are the parameters of the zero-shot model.

tuned solution while decreasing performance on the reference distribution by at most 0.3 pp (accuracy on the reference distribution often improves). In contrast to the ImageNet distribution shifts, the zero-shot model initially achieves less than 30% accuracy on the WILDS distribution shifts, and WiSE-FT provides improvements regardless.

Hyperparameter variation and alternatives. As illustrated by Figure 2.3, moderate changes in standard hyperparameters such as the learning rate or the number of epochs can substantially affect performance under distribution shift. Moreover, these performance differences cannot be detected reliably from model performance on reference data alone. For instance, while training for 10 epochs with learning rate $3 \cdot 10^{-5}$ and $3 \cdot 10^{-6}$ lead to a small accuracy difference on ImageNet (0.3 pp), accuracy under distribution shift

	ImageNet	CIFAR10	CIFAR100	Cars	DTD	SUN397	Food101
Standard fine-tuning	86.2	98.6	92.2	91.6	81.9	80.7	94.4
WiSE-FT (α=0.5)	86.8 (+0.6)	99.3 (+0.7)	93.3 (+1.1)	93.3 (+1.7)	84.6 (+2.8)	83.2 (+2.5)	96.1 (+1.6)
WiSE-FT (opt. α)	87.1 (+0.9)	99.5 (+0.8)	93.4 (+1.2)	93.6 (+2.0)	85.2 (+3.3)	83.3 (+2.6)	96.2 (+1.8)

Table 2.2: Beyond robustness, WiSE-FT can improve accuracy after fine-tuning on several datasets.

varies by as much as 8 pp.

Furthermore, tuning hyperparameters on ImageNet data can also reduce robustness. For instance, while moving from small to moderate learning rates (10^{-7} to $3 \cdot 10^{-5}$) improves performance on ImageNet by 5 pp, it also deteriorates accuracy under distribution shift by 8 pp.

WiSE-FT addresses this brittleness of hyperparameter tuning: even when using a learning rate $3 \cdot 10^{-5}$ where standard fine-tuning leads to low robustness, applying WiSE-FT removes the trade-off between accuracy on the reference and shifted distributions. The models which can be achieved by varying α are as good or better than those achievable by other hyperparameter configurations. Then, instead of searching over a wide range of hyperparameters, only α needs to be considered. Moreover, evaluating different values of α does not require training new models.

There is no hyperparameter in Figure 2.3 which can be varied to match or exceed the optimal curve produced by WiSE-FT. In our experiments, this frontier is reached only through methods that average model weights, either using WiSE-FT or with a more sophisticated averaging scheme: keeping an exponential moving average of all model iterates (EMA, [249]).

Accuracy gains on reference distributions. Beyond robustness to distribution shift, Table 2.2 demonstrates that WiSE-FT also improves accuracy after fine-tuning on seven datasets. When fine-tuning end-to-end on ImageNet, CIFAR-10, CIFAR-100, Describable Textures, Food-101, SUN397, and Stanford Cars, WiSE-FT reduces relative error by 4 to 49%. Even though standard fine-tuning directly optimizes for high accuracy on the reference distribution, WiSE-FT achieves better performance.

Beyond CLIP. Figure 2.4 illustrates that WiSE-FT is generally applicable to zero-shot models beyond CLIP, and beyond models pre-trained contrastively with image-text pairs. First, we interpolate between the weights of the zero-shot and fine-tuned BASIC-L model [202], finding that α=0.5 improves average

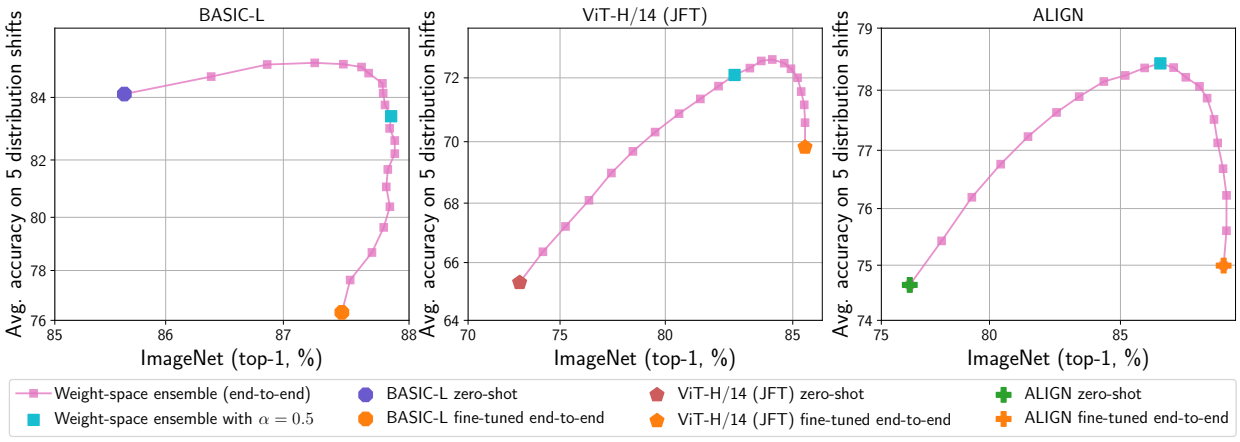

Figure 2.4: WiSE-FT applied to BASIC-L [202], a ViT-H/14 [71] model pre-trained on JFT-300M [245] and ALIGN [125].

accuracy on five distribution shifts derived from ImageNet by over 7 pp while improving ImageNet accuracy by 0.4 pp relative to the fine-tuned BASIC-L model. As in Pham et al. [202], the model is fine-tuned using a contrastive loss and half of the ImageNet training data. WiSE-FT provides improvements on both reference and shifted distributions, despite these experimental differences.

Next, we consider the application of WiSE-FT to a ViT-H/14 model [71] pre-trained on JFT-300M [245], where the zero-shot classifier is constructed by manually identifying a class correspondence. WiSE-FT improves performance under distribution shift over both the zero-shot and fine-tuned models. When α=0.8, WiSE-FT outperforms the fine-tuned model by 2.2 pp on distribution shifts, while maintaining ImageNet performance within 0.2 pp of the fine-tuned model. This result demonstrates that WiSE-FT can be successfully applied even to models which do not use contrastive image-text pre-training.

Finally, we apply WiSE-FT to the ALIGN model of Jia et al. [125], which is similar to CLIP but is pre-trained with a different dataset, finding similar trends.

2.6 Discussion

This section further analyzes the empirical phenomena we have observed so far. We begin with the case where only the final linear layer is fine-tuned and predictions from the weight-space ensemble can be factored into the outputs of the zero-shot and fine-tuned model. Next, we connect our observations regarding end-to-end fine-tuning with earlier work on the phenomenology of deep learning.

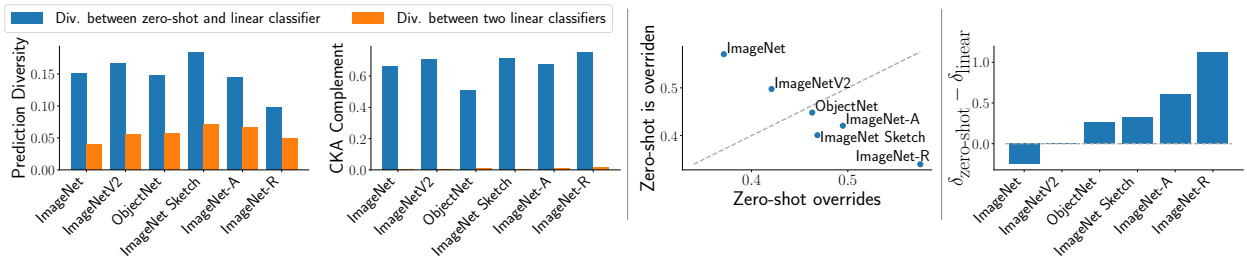

Figure 2.5: (Left) Zero-shot and fine-tuned models exhibit diversity in their predictions. **(Middle)** On most distribution shifts, the zero-shot model overrides the linear classifier more than it is overridden. The reverse is true for ImageNet (reference). **(Right)** Similarly, zero-shot models are more confident under distribution shift, while the reverse is true on the reference distribution. The margin δ_f measures the average difference between the largest and second largest unormalized output for classifier f

2.6.1 Zero-shot and fine-tuned models are complementary

In this section, we find that the zero-shot and fine-tuned models have diverse predictions, both on reference and shifted distributions. Moreover, while the fine-tuned models are more confident on the reference distribution, the reverse is true under distribution shift.

Zero-shot and fine-tuned models are diverse. In certain cases, ensemble accuracy is correlated with diversity among the constituents [150, 101]. If two models make coincident mistakes, so will their ensemble, and no benefit will be gained from combining them. Here, we explore two measures of diversity: *prediction diversity,* which measures the fraction of examples for which two classifiers disagree but one is correct; and *Centered Kernel Alignment Complement,* the complement of CKA [141]. In Figure 2.5 (left), we show that the zero-shot and fine-tuned models are diverse both on the reference and shifted distributions, despite sharing the same backbone. As a point of comparison, we include avg. diversity measures between two linear classifiers fine-tuned with random splits on half of ImageNet,[3] denoted in orange in Figure 2.5.

Models are more confident where they excel. In order for the ensemble model to be effective, it should leverage each model's expertise based on which distribution the data is from. Here, we empirically show that this occurs on a number of datasets we consider. First, we examine the cases where the models being ensembled disagree. We say the zero-shot model *overrides* the fine-tuned model if their predictions disagree

[3]Two linear classifiers fine-tuned on the same data converge to similar solutions, resulting in negligible diversity. As a stronger baseline, we fine-tune classifiers on different subsets of ImageNet, with half of the data.

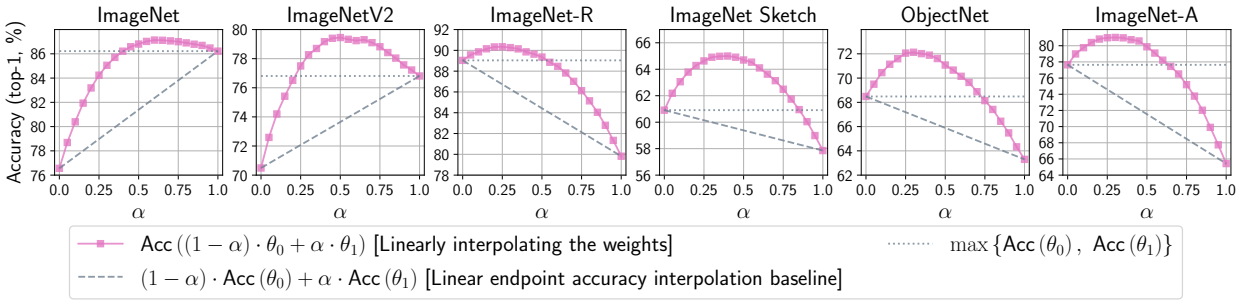

Figure 2.6: On ImageNet and the main distribution shifts we consider, linearly interpolating between the weights of θ_0 and θ_1 exceeds the baseline of linearly interpolating the accuracies of the two models for all α (Observation 1). Moreover, there exists an α for which WiSE-FT outperforms both the zero-shot and fine-tuned models (Observation 2).

and the zero-shot prediction matches that of the weight-space ensemble. Similarly, if models disagree and the linear classifier prediction matches the ensemble, we say the zero-shot is *overridden*. Figure 2.5 (middle) shows the fraction of samples where the zero-shot model overrides and is overridden by the fine-tuned linear classifier for α=0.5. Other than ImageNetV2, which was collected to closely reproduce ImageNet, the zero-shot model overrides the linear classifier more than it is overridden on the distribution shifts.

Additionally, we are interested in measuring model confidence. Recall that we are ensembling quantities before a softmax is applied, so we avoid criteria that use probability vectors, e.g., Guo et al. [103]. Instead, we consider the margin δ between the largest and second largest output of each classifier. Figure 2.5 (right) shows that the zero-shot model is more confident in its predictions under distribution shift, while the reverse is true on the reference distribution.

2.6.2 An error landscape perspective

We now turn to empirical phenomena we observe when weight-space ensembling *all* layers in the network. Specifically, this section formalizes our observations and details related phenomena. Recall that the weight-space ensemble of θ_0 and θ_1 is given by $f(x, (1-\alpha)\cdot\theta_0 + \alpha\cdot\theta_1)$ (Equation 2.1).

For a distribution $\mathcal{D}$ and model f, let $\text{Acc}_{\mathcal{D},f}(\theta)$ denote the expected accuracy of f evaluated with parameters θ on distribution $\mathcal{D}$.

Observation 1: As illustrated in Figure 2.6, on ImageNet and the five associated distribution shifts we

consider

$$\mathsf{Acc}_{\mathcal{D},f}((1-\alpha)\cdot\theta_0+\alpha\cdot\theta_1)\ \geq\ (1-\alpha)\cdot\mathsf{Acc}_{\mathcal{D},f}(\theta_0)+\alpha\cdot\mathsf{Acc}_{\mathcal{D},f}(\theta_1) \tag{2.2}$$

for all $\alpha\in[0,1]$.

Note that equation 2.2 uses the baseline of linearly interpolating between the accuracies of the two endpoints, which is always achievable by using weights θ_1 with probability α and using model θ_0 otherwise. In the case where the accuracy of both endpoints are similar, Equation 2.2 is equivalent to the definition of Linear Mode Connectivity of Frankle et al. [82].

To assist in contextualizing Observation 1, we review related phenomena. Neural networks are non-linear, hence weight-space ensembles only achieve good performance in exceptional cases—interpolating the weights of two networks trained on ImageNet from the same random initialization results in no better accuracy than a random classifier [82]. On the simpler MNIST task [152], linear mode connectivity was observed by Nagarajan and Kolter [190] between a pair of models trained from the same random initialization. Linear mode connectivity has also been observed for harder tasks such as ImageNet by Frankle et al. [82]; Izmailov et al. [123] when part of the training trajectory is shared. Finally, Neyshabur et al. [192] observe linear mode connectivity between two models that are fine-tuned from a shared, pre-trained initialization. In particular, the observations of Neyshabur et al. [192] may elucidate why weight-space ensembles attain high accuracy in the setting we consider, as they suggest that fine-tuning remains in a region where solutions are connected by a linear path along which error remains low. Instead of considering the weight-space ensemble of two fine-tuned models, we consider the weight-space ensemble of the *pre-trained* and fine-tuned models. This is only possible for a pre-trained model capable of zero-shot inference such as CLIP.

Observation 2: As illustrated by Figure 2.6, on ImageNet and the five associated distribution shifts we consider, weight-space ensembling (end-to-end) may outperform both the zero-shot and fine-tuned models, i.e., there exists an α for which $\mathsf{Acc}_{\mathcal{D},f}\left((1-\alpha)\cdot\theta_0+\alpha\cdot\theta_1\right)\ \geq\ \max\left\{\mathsf{Acc}_{\mathcal{D},f}\left(\theta_0\right),\ \mathsf{Acc}_{\mathcal{D},f}\left(\theta_1\right)\right\}$.

We are not the first to observe that when interpolating between models, the accuracy of models along the path may exceed that of either endpoint [123, 192, 270]. Neyshabur et al. [192] conjecture that interpolation could produce solutions closer to the true center of a basin. In contrast to Neyshabur et al. [192], we interpolate between models which observe different data.

2.7 Related work

Robustness. Understanding how models perform under distribution shift remains an important goal, as real world models may encounter data from new environments [207, 253]. Previous work has studied model behavior under synthetic [110, 256, 174, 92, 77, 5] and natural distribution shift [112, 138, 260, 17, 113]. Interventions used for synthetic shifts do not typically provide robustness to many natural distribution shifts [250]. In contrast, accuracy on the reference distribution is often a reliable predictor for accuracy under distribution shift [279, 185, 250, 246, 186]. On the other hand, D'Amour et al. [55] show that accuracy under certain distribution shifts cannot be reliably inferred from accuracy on the reference distribution. We observe a similar phenomenon when fine-tuning with different hyperparameters (Section 2.5, Figure 2.3).

Pre-training and transfer learning. Pre-training on large amounts of data is a powerful technique for building high-performing machine learning systems [233, 71, 139, 281, 209, 33]. One increasingly popular class of vision models are those pre-trained with auxiliary language supervision, which can be used for zero-shot inference [60, 229, 302, 210, 125, 202, 293]. When pre-trained models are adapted to a specific distribution through standard fine-tuning, effective robustness deteriorates at convergence [6]. In natural language processing, previous work proposed stable fine-tuning methods that incur computational overhead [126, 305], alleviating problems such as representational collapse [1]. More generally, a variety of methods have attempted to mitigate catastrophic forgetting [178]. Kirkpatrick et al. [137]; Zenke et al. [290] explored weighted quadratic regularization for sequential learning. Xuhong et al. [278] showed that, for fine-tuning, the simple quadratic regularization explored in Section 2.5 performs best, while Lubana et al. [172] explored the connection between quadratic regularization and interpolation. Andreassen et al. [6] found that many approaches from continual learning do not provide robustness to multiple natural distribution shifts. Finally, Li et al. [159] investigate the effect of fine-tuning hyperparameters on performance.

Traditional (output-space) ensembles. Traditional ensemble methods, which we refer to as output-space ensembles, combine the predictions (outputs) of many classifiers [66, 18, 31, 85, 151, 84]. Typically, output-space ensembles outperform individual classifiers and provide uncertainty estimates under distribution shift that are more callibrated than baselines [151, 198, 243]. In contrast to these works, we consider the ensemble

of two models which have observed different data. Output-space ensembles require more computational resources as they require a separate pass through each model. Compared to an ensemble of 15 models trained on the same dataset, Mustafa et al. [189] find an improvement of 0.8–1.6 pp under distribution shift (on ImageNetV2, ImageNet-R, ObjectNet, and ImageNet-A) by ensembling a similar number of models pre-trained on different datasets. In contrast, we see an improvement of 2–15 pp from ensembling two models. Moreover, as we ensemble in weight-space, no extra compute is required compared to a single model.

Weight-space ensembles. Weight-space ensembles linearly interpolate between the weights of different models [190, 82, 173, 102, 249]. For example, Izmailov et al. [123] average checkpoints saved throughout training for improved performance. Indeed, averaging the weights along the training trajectory is a central method in optimization [227, 204, 194]. For instance, Zhang et al. [298] propose optimizing with a set of fast and slow weights, where every k steps, these two sets of weights are averaged and a new trajectory begins. Here, we revisit these techniques from a distributional robustness perspective and consider the weight-space ensemble of models which have observed different data.

Concurrent and subsequent work. Topics including robust fine-tuning, ensembles for improved robustness, and interpolating the weights of fine-tuned models are studied in concurrent and subsequent work. Kumar et al. [148] observe that fine-tuning end-to-end often results in higher accuracy on the reference distribution but lower accuracy under distribution shift, compared to linear classifier fine-tuning. To address this, Kumar et al. [148] first fine-tune a linear classifier and use this as the initialization for end-to-end fine-tuning. We consider fine-tuning zero-shot models, and so we begin with a classifier (i.e., the zero-shot classifier) which we are using as the initialization for end-to-end fine-tuning. In a separate work, Kumar et al. [146] find that calibrated output-space ensembles can be used to mitigate accuracy trade-offs.

Hewitt et al. [114] explore the application of output-space ensembles and distillation to mitigate accuracy trade-offs which arise in fine-tuning models for natural language generation. Gontijo-Lopes et al. [100] explore output-space ensembles of models across hyper-parameters, architectures, frameworks, and datasets. They find that specializing in subdomains of data leads to high ensemble performance. Finally, Matena and Raffel [176] introduce a method of combining models in weight-space that goes beyond linear interpolation

with a single mixing-coefficient as employed in WiSE-FT. Specifically, Matena and Raffel [176] employ Fisher information as a measure of per-parameter importance. While their experiments do not examine accuracy under distribution shift, their goal of combining differing expertise into one shared model is well aligned with ours.

2.8 Limitations, impact, and conclusion

Limitations. While we expect our findings to be more broadly applicable to other domains such as natural language processing, our investigation here is limited to image classification. Exploring fine-tuning for object detection and natural language processing are interesting directions for future work. Moreover, although the interpolation parameter setting α=0.5 provides good overall performance, we leave the question of finding the optimal α for specific target distributions to future work.

Impact. Radford et al. [210] and Brown et al. [33] extensively discuss the broader impact of large zero-shot models and identify potential causes of harm including model biases and potential malicious uses such as surveillance systems. WiSE-FT is a fine-tuning method that builds on such models, and thus may perpetuate their negative impact.

Conclusion. WiSE-FT can substantially improve performance under distribution shift with minimal or no loss in accuracy on the target distribution compared to standard fine-tuning. We view WiSE-FT as a first step towards more sophisticated fine-tuning schemes and anticipate that future work will continue to leverage the robustness of zero-shot models for building more reliable neural networks.

Chapter 3

Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

3.1 Overview

The conventional recipe for maximizing model accuracy is to (1) train multiple models with various hyperparameters and (2) pick the individual model which performs best on a held-out validation set, discarding the remainder. In this paper, we revisit the second step of this procedure in the context of fine-tuning large pre-trained models, where fine-tuned models often appear to lie in a single low error basin. We show that averaging the weights of multiple models fine-tuned with different hyperparameter configurations often improves accuracy and robustness. Unlike a conventional ensemble, we may average many models without incurring any additional inference or memory costs—we call the results "model soups." When fine-tuning large pre-trained models such as CLIP, ALIGN, and a ViT-G pre-trained on JFT, our soup recipe provides significant improvements over the best model in a hyperparameter sweep on ImageNet. The resulting ViT-G model, which attains 90.94% top-1 accuracy on ImageNet, achieved a new state of the art. Furthermore, we show that the model soup approach extends to multiple image classification and natural language processing tasks, improves out-of-distribution performance, and improves zero-shot performance on new downstream

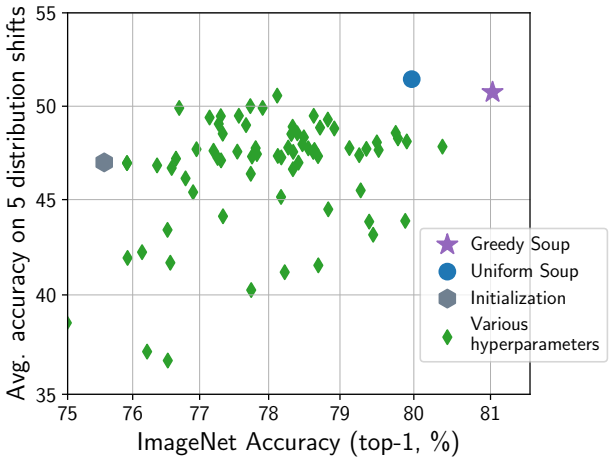

Figure 3.1: *Model soups* improve accuracy over the best individual model when performing a large, random hyperparameter search for fine-tuning a CLIP ViT-B/32 model on ImageNet. The *uniform soup* (blue circle) averages all fine-tuned models (green diamonds) in a random hyperparameter search over learning rate, weight-decay, iterations, data augmentation, mixup, and label smoothing. The *greedy soup* adds models sequentially to the model soup, keeping a model in the soup if accuracy on the held-out validation set does not decrease.

tasks. Finally, we analytically relate the performance similarity of weight-averaging and logit-ensembling to flatness of the loss and confidence of the predictions, and validate this relation empirically. Code is available at https://github.com/mlfoundations/model-soups.

3.2 Introduction

In recent years, research has shown that models pre-trained on large and diverse datasets learn representations that transfer well to a variety of tasks. As a result, machine learning practitioners now commonly develop solutions for downstream tasks by fine-tuning large pre-trained models [97, 285, 142, 139]. Typically, the fine-tuning process involves two steps: (1) fine-tune models with a variety of hyperparameter configurations, and (2) select the model which achieves the highest accuracy on the held-out validation set. The remaining models are then discarded.

Selecting a single model and discarding the rest has several downsides. For one, ensembling outputs of many models can outperform the best single model, albeit at a high computational cost during inference. For another, fine-tuning a model on downstream tasks can sometimes reduce out-of-distribution performance [210, 6, 272, 202], and the best single model on the target distribution may not be the best model on out-of-

Method	ImageNet acc. (top-1, %)	Distribution shifts
ViT-G [292]	90.45	–
CoAtNet-7 [53]	90.88	–
Our models/evaluations based on ViT-G:		
ViT-G (reevaluated)	90.47	82.06
Best model in hyperparam search	90.78	84.68
Greedy soup	**90.94**	**85.02**

Table 3.1: *Model soups* improve accuracy over the best individual model when fine-tuning a JFT-3B pre-trained ViT-G/14 model on ImageNet. Instead of selecting the best model from a hyperparameter sweep during fine-tuning, *model soups* average the weights of multiple fine-tuned models. To evaluate performance under distribution shift we consider average accuracy on ImageNet-V2, ImageNet-R, ImageNet-Sketch, ObjectNet, and ImageNet-A. Additional details are provided by Table 3.4 and Section 3.4.3.

distribution data.

In this work, we propose a more accurate and robust alternative to the second step of the conventional recipe in the context of fine-tuning a large pre-trained model. Instead of selecting the individual fine-tuned model which achieves the highest accuracy on the held-out validation set, we average the weights of models fine-tuned independently, and refer to the result as a *model soup*. Given the results of the first step—a hyperparameter sweep over fine-tuned models—averaging several of these models to form a model soup requires no additional training and adds no cost at inference time.

Since the loss landscape of neural network training is non-convex with many solutions in different loss basins, it is perhaps surprising that averaging the weights of independently fine-tuned models achieves high performance. However, recent work [192] observes that fine-tuned models optimized independently from the same pre-trained initialization lie in the same basin of the error landscape, inspiring our method. Weight averaging along a single training trajectory has previously been shown to improve the performance of models in non-transfer settings [249, 123]. Our approach extends weight averaging to the context of fine-tuning, where we find that it also works across many independent runs with varied hyperparemeter configurations. Our use of a diverse set of fine-tuned models is inspired by Gontijo-Lopes et al. [101] who observe that ensembling independent runs trained with different hyperparameters improves performance.

We perform a comprehensive experimental study of fine-tuning to understand the behavior of model soups. For our main results we fine-tune CLIP [210] and ALIGN [125], which are pre-trained with a contrastive loss on image-text pairs, and a ViT-G model pre-trained on JFT [292]. Our results show that

model soups often outperform the best individual model on both the in-distribution and natural distribution shift test sets (Table 3.1, Figure 3.1, Figure 3.5). A model soup composed of ViT-G models achieves 90.94% on ImageNet [59], surpassing the previous state of the art of 90.88% attained by the CoAtNet model [53] while requiring 25% fewer FLOPs at inference time.[1] In general, model soups can approach the performance of ensembling, with no additional computational cost or memory relative to a single model during inference. Beyond ImageNet and associated distribution shifts, our results show that model soups are applicable when fine-tuning on tasks from the WILDS [138] benchmark, and when fine-tuning transformer models [257, 64, 215] for text classification.

While the most straightforward approach to making a model soup is to average all the weights uniformly, we find that *greedy soups*, where models are sequentially added to the soup if they improve accuracy on held-out data, outperforms uniform averaging. Greedy soups avoid adding in models which may lie in a different basin of the error landscape, which could happen if, for example, models are fine-tuned with high learning rates.

In addition to empirical observations, we analytically relate the similarity in loss between weight-averaging and logit-ensembling to the flatness of the loss (i.e., its second derivative on a line between models) and confidence of the predictions (expressed via the variance of a logits difference drawn from the weight-average softmax). We empirically validate our approximation on a subset of the models we train and show that it is strongly correlated with the true averaging vs. ensembling performance difference, particularly in the learning rate regimes where soups are effective and models achieve higher accuracy.

Paper outline. Our method of *model soups* is presented and evaluated in Sections 3.3 and 3.4, respectively. Next, Section 3.5 includes our analysis relating model soups and ensembles, Section 3.6 details the scope and limitations of the proposed method, and Section 3.7 contextualizes *model soups* by reviewing related work.

Table 3.2: The primary methods contrasted in this work. Each θ_i is a model found through fine-tuning from a shared initialization. Cost refers to the memory and compute requirements during inference relative to a single model. All methods require the same training.

	Method	Cost
Best on val. set	$f\left(x, \text{argmax}_i \text{ValAcc}\left(\theta_i\right)\right)$	$\mathcal{O}(1)$
Ensemble	$\frac{1}{k}\sum_{i=1}^{k} f(x, \theta_i)$	$\mathcal{O}(k)$
Uniform soup	$f\left(x, \frac{1}{k}\sum_{i=1}^{k} \theta_i\right)$	$\mathcal{O}(1)$
Greedy soup	Recipe 1	$\mathcal{O}(1)$
Learned soup	Appendix	$\mathcal{O}(1)$

3.3 Method

This section highlights three recipes for model souping, the *uniform*, *greedy*, and *learned* soup, though the greedy soup is our central method. We summarize the methods described in this section in Table 3.2.

We consider a neural network $f(x, \theta)$ with input data x and parameters $\theta \in \mathbb{R}^d$. Fine-tuning is analogous to standard neural network training but includes an important distinction: the parameters are initialized to those found via pre-training.

Let $\theta = \text{FineTune}(\theta_0, h)$ denote the parameters obtained by fine-tuning with pre-trained initialization θ_0 and hyperparameter configuration h. The hyperparameter configuration can include the choice of optimizer, data augmentation, training iterations, and a random seed which will determine data order.

For hyperparameter configurations $h_1, ..., h_k$ let $\theta_i = \text{FineTune}(\theta_0, h_i)$. Conventionally, the parameters θ_j which attain the highest accuracy on a held out validation set are selected, and the remaining parameters are discarded. Instead, *model soups* $f(x, \theta_{\mathcal{S}})$ use an average of θ_i, i.e., $\theta_{\mathcal{S}} = \frac{1}{|\mathcal{S}|}\sum_{i \in \mathcal{S}} \theta_i$ where $\mathcal{S} \subseteq \{1, ..., k\}$. The *uniform soup* is constructed by averaging *all* fine-tuned models θ_i and so $\mathcal{S} = \{1, ..., n\}$.

There are settings in which a hyperparameter configuration can produce a model with low accuracy that results in a low accuracy uniform soup. This issue can be circumvented with a *greedy soup* (Recipe 1). The greedy soup is constructed by sequentially adding each model as a potential ingredient in the soup, and only keeping the model in the soup if performance on a held out validation set (disjoint from the training and test sets) improves. Before running this procedure we sort the models in decreasing order of validation set

[1]Since our initial submission, we attain 90.98% with BASIC [202], which ties the newer CoCa model [287] to their reported precision.

Algorithm 1 GreedySoup

Input: Potential soup ingredients $\{\theta_1, ..., \theta_k\}$ (sorted in decreasing order of ValAcc(θ_i)).
ingredients $\leftarrow \{\}$
for $i = 1$ **to** k **do**
  if ValAcc(average(ingredients $\cup \{\theta_i\}$)) $\geq$
      ValAcc(average(ingredients)) **then**
    ingredients $\leftarrow$ ingredients $\cup \{\theta_i\}$
return average(ingredients)

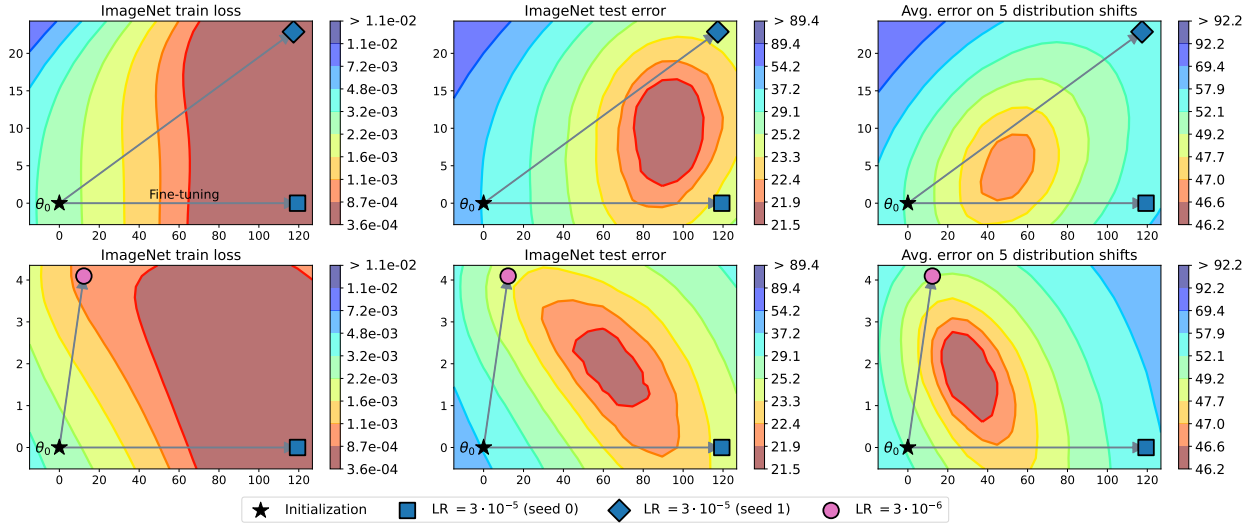

Figure 3.2: The solution with the highest accuracy is often not a fine-tuned model but rather lies between fine-tuned models. This figure shows loss and error on a two dimensional slice of the loss and error landscapes. We use the zero-shot initialization θ_0 and fine-tune twice (illustrated by the gray arrows), independently, to obtain solutions θ_1 and θ_2. As in Garipov et al. [90], we obtain an orthonormal basis u_1, u_2 for the plane spanned by these models, and the x and y-axis show movement in parameter space in these directions, respectively.

accuracy, and so the greedy soup can be no worse than the best individual model on the held-out validation set. We also explore a more advanced *learned soup* recipe that optimizes model interpolation weights by gradient-based minibatch optimization. This procedure requires simultaneously loading all models in memory which currently hinders its use with large networks.

3.4 Experiments

This section presents our key experimental findings. We begin with experimental setup (Section 3.4.1) then provide intuition for model soups by examining error landscape visualizations (Section 3.4.2). Next we

present our main results (Section 3.4.3), using model soups as an alternative to selecting the best performing individual model. The appendix includes additional results on model soups in the context of robust fine-tuning.

3.4.1 Experimental setup

Our experiments explore the application of model soups when fine-tuning various models. The primary models we fine-tune are the CLIP [210], ALIGN [125], and BASIC [202] models pre-trained with contrastive supervision from image-text pairs, a ViT-G/14 model pre-trained on JFT-3B [292], and transformer models for text classification [64, 214]. Unless otherwise mentioned, experiments use the CLIP ViT-B/32 model. Fine-tuning is performed end-to-end (all parameters are modified) which typically results in better accuracy than training only the final linear layer [142, 2, 36, 10].

We consider two different methods for initializing the final linear layer before fine-tuning. The first method initializes the model from a linear probe (LP), as described in Kumar et al. [148], and we refer to this method as LP initialization. The second method uses the zero-shot initialization, e.g., using the classifier produced by the text tower of CLIP or ALIGN as the initialization. Both methods for initializing the model produce similar trends when applicable, and unless otherwise stated we use the LP initialization.

For the ensemble baselines [66, 151] we ensemble the logits (unormalized outputs) of models as in Gontijo-Lopes et al. [101]. Fine-tuning uses a supervised cross-entropy loss and, unless otherwise mentioned, is conducted on ImageNet [59]. When fine-tuning on ImageNet we also evaluate on the five natural distribution shifts: ImageNetV2 [222], ImageNet-R [112], ImageNet-Sketch [260], ObjectNet [17], and ImageNet-A [113]. We often report results averaged over these five distribution shifts. Since the official ImageNet validation set is typically used as the test set, we use roughly 2% of the ImageNet training set as a held-out validation set for constructing greedy soups.

3.4.2 Intuition and motivation

Error landscape visualizations. To provide intuition, we visualize a two dimensional slice of the training loss and test error landscape when fine-tuning CLIP on ImageNet. In these experiments, we use the zero-shot initialization $\theta_0 \in \mathbb{R}^d$ and fine-tune twice, independently, to produce solutions θ_1 and θ_2. The points

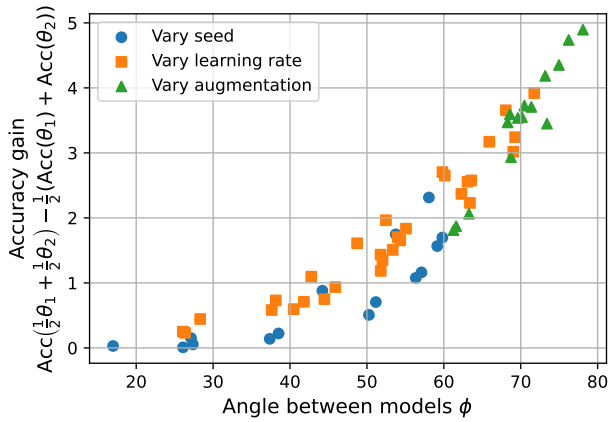

Figure 3.3: The advantage of averaging solutions (y-axis) is correlated with the angle ϕ between between solutions, while varying hyperparameter configurations between pairs enables a larger ϕ. Each point corresponds to a pair of models θ_1, θ_2 that are fine-tuned independently from a shared initialization θ_0 with different hyperparameter configurations. The angle ϕ between between solutions refers to the angle between $\theta_1 - \theta_0$ and $\theta_2 - \theta_0$ (i.e., the initialization is treated as the origin). Accuracy is averaged over ImageNet and the five distribution shifts described in Section 3.4.1.

θ_0, θ_1 and θ_2 define a plane in parameter space, and we evaluate the ImageNet train loss, ImageNet test error, and the test error on the five aforementioned distribution shifts on this plane. The results are illustrated in Figure 3.2 where the zero-shot initialization (θ_0) is shown as a star and a solution fine-tuned with learning rate $3 \cdot 10^{-5}$ (θ_1) is shown as a blue square. For θ_2 we either use the same learning rate as θ_1 (but vary the random seed) or learning rate $3 \cdot 10^{-6}$. For both the in-distribution and out-of-distribution test sets, the loss/error contours are basin-shaped, and none of the three points is optimal.

These results suggest that (1) interpolating the weights of two fine-tuned solutions can improve accuracy compared to individual models and (2) more uncorrelated solutions—models that form an angle[2] closer to 90 degrees—may lead to higher accuracy on the linear interpolation path.

To investigate the correlation between accuracy improvement and angle, we consider a series of models trained with different seeds, learning rates, and data augmentation. For each pair θ_1, θ_2, we compare the accuracy of their average with the average of their accuracies, $\mathrm{Acc}\left(\frac{1}{2}\theta_1 + \frac{1}{2}\theta_2\right) - \frac{1}{2}\left(\mathrm{Acc}\left(\theta_1\right) + \mathrm{Acc}\left(\theta_2\right)\right)$, which we refer to as the interpolation advantage. Figure 3.3 illustrates the results, in which we observe that the interpolation advantage is correlated with the angle ϕ and that varying the learning rate, seed, or data augmentation can produce solutions which are more orthogonal.

[2]In particular, the angle ϕ between $\theta_1 - \theta_0$ and $\theta_2 - \theta_0$, i.e., the angle between the arrows shown in Figure 3.2.

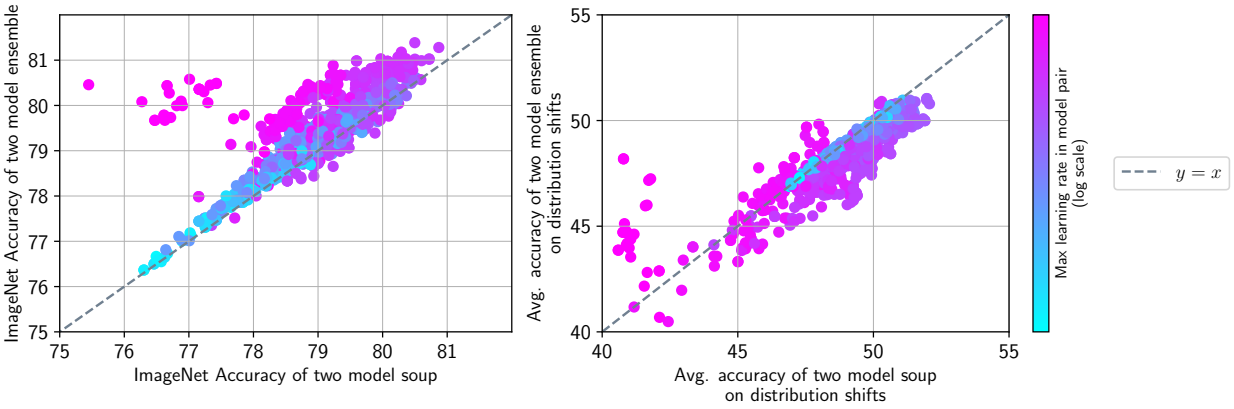

Figure 3.4: Ensemble performance is correlated with model soup performance. Each point on the scatter plot is a model pair with different hyperparameters. The x-axis is the accuracy when the weights of the two models are averaged (i.e., the two model soup) while the y-axis is the accuracy of the two model ensemble. Ensembles often perform slightly better than soups on ImageNet (left) while the reverse is true on the distribution shifts (right). Each model pair consists of two random greed diamonds from Figure 3.1.

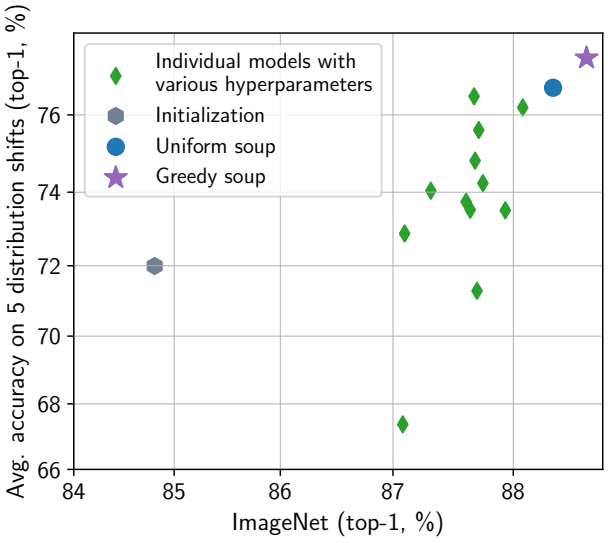

Figure 3.5: *Model soups* improve accuracy when fine-tuning ALIGN.

Ensemble comparison. Figure 3.4 observes that ensemble performance is correlated with soup performance for moderate and small learning rates. We consider pairs of models selected at random from the individual solutions in Figure 3.1, and find that the maximum learning rate of the models in the pair is indicative of the ensemble accuracy, soup accuracy, and their relation: When learning rate is small, ensemble accuracy and soup accuracy are similar, but both are suboptimal. For moderate learning rate values, ensemble accuracy and soup accuracy are both high. For high learning rate values, ensemble performance exceeds soup performance, but ensembles/soups with moderate learning rates perform better. Overall, ensembles achieve higher accuracy on ImageNet while the reverse is true on the distribution shifts.

3.4.3 Model soups

With the gains of averaging two fine-tuned models in mind, we turn our attention to averaging *many* models with different hyperparameters: this section presents our main results, which show that averaging fine-tuned models can be used as an alternative to the conventional procedure of selecting the single model which performs best on the held-out validation set. We explore CLIP [210] and ALIGN [125] fine-tuned on ImageNet [59] (Section 3.4.3), ViT-G pre-trained on JFT-3B [292] and fine-tuned on ImageNet (Section 3.4.3), and transformer models fine-tuned on text classification tasks (Section 3.4.3).

Fine-tuning CLIP and ALIGN

We begin our study of model soups by considering two-pretrained models, CLIP ViT-B/32 and ALIGN EfficientNet-L2, and performing a hyperparameter sweep for the fine-tuning each model on ImageNet. For CLIP we use a random hyperparameter search over learning rate, weight decay, training epochs, label smoothing, and data augmentation, obtaining 72 fine-tuned models. For ALIGN we use a grid search over learning rate, data augmentation, and mixup, obtaining 12 fine-tuned models. To form our greedy soups, we sort models in order of decreasing accuracy on the held-out validation set before applying Recipe 1. For both CLIP and ALIGN, the greedy soup selects 5 models. Figure 3.1 and 3.5 show the performance of the resulting models and their uniform and greedy soups for CLIP and ALIGN. The greedy soup improves on over the best model in the hyperparameter sweep by 0.7 and 0.5 percentage points, respectively.

Furthermore, we show that, for essentially any number of models, the greedy soup outperforms the best

Table 3.3: Ablation on multiple methods from Table 3.2 and their variants when when fine-tuning CLIP ViT-B/32 with the random hyperparameter search described in Section 3.4.3. For "Greedy soup (random order)", we try three random model orders when running the greedy soup procedure (by default, models are sorted by decreasing held-out val accuracy). The "Learned soup" and its variants are descried in the Appendix. The *best* in *best individual model* refers to ImageNet accuracy.

	ImageNet	Dist. shifts
Best individual model	80.38	47.83
Second best model	79.89	43.87
Uniform soup	79.97	51.45
Greedy soup	81.03	50.75
Greedy soup (random order)	80.79 (0.05)	51.30 (0.16)
Learned soup	80.89	51.07
Learned soup (by layer)	81.37	50.87
Ensemble	81.19	50.77
Greedy ensemble	81.90	49.44

single model on both the ImageNet and the out-of-distribution test sets. We consider an additional setting where we prepare a sequence of soups by sequentially adding CLIP models from the hyperparameter sweep in random order. The Appendix shows the performance of the uniform and greedy soup, as well as the best single model so far and a logit ensemble, as a function of the number of models considered. The greedy soup is better than the uniform soup on ImageNet and comparable to it out-of-distribution. The logit ensemble is better than the greedy soup on ImageNet, but worse out-of-distribution.

Table 3.3 lists the performance of the CLIP soups and baselines described above, as well as additional soup variants.

To further establish the generality of the model soup, we replicate the CLIP hyperparameter sweep experiment on two image classification tasks from WILDS [138], namely FMoW [45] and iWildCam [19].

Fine-tuning a ViT-G model pre-trained on JFT-3B

To test whether the gains obtained by model soups are additive with other techniques used to obtain state-of-the-art models, we applied our greedy soup technique to 58 ViT-G/14 models fine-tuned on ImageNet. We vary the learning rate, decay schedule, loss function, and minimum crop size in the data augmentation, and optionally apply RandAugment [50], mixup [296], or CutMix [289]. We also train four models with sharpness-aware minimization (SAM) [80]. For each model training run, we save exponential moving

Table 3.4: Greedy soup improves over the best individual models obtained in a hyperparameter sweep for ViT-G/14 pre-trained on JFT-3B and fine-tuned on ImageNet, both in- and out-of-distribution. Accuracy numbers not significantly different from the best are bold-faced. Statistical comparisons are performed using an exact McNemar test or permutation test at $\alpha = 0.05$. Avg shift accuracy of the best model on each test set is the best average accuracy of any individual model.

	ImageNet			Distribution shifts					
Method	Top-1	ReaL	Multilabel	IN-V2	IN-R	IN-Sketch	ObjectNet	IN-A	Avg shifts
ViT/G-14 [292]	90.45	90.81	–	83.33	–	–	70.53	–	–
CoAtNet-7 [53]	90.88	–	–	–	–	–	–	–	–
Our models/evaluations based on ViT-G/14:									
ViT/G-14 [292] (reevaluated)	90.47	90.86	96.89	83.39	94.38	72.37	71.16	89.00	82.06
Best model on held out val set	90.72	91.04	96.94	83.76	95.04	73.16	78.20	91.75	84.38
Best model on each test set (oracle)	90.78	**91.78**	**97.29**	**84.31**	95.04	73.73	**79.03**	92.16	84.68
Greedy ensemble	**90.93**	91.29	**97.23**	**84.14**	94.85	73.07	77.87	91.69	84.33
Greedy soup	**90.94**	91.20	**97.17**	**84.22**	**95.46**	**74.23**	78.52	**92.67**	**85.02**

Table 3.5: Performance of model soups on four text classification datasets from the GLUE benchmark [259].

Model	Method	MRPC	RTE	CoLA	SST-2
BERT [65]	Best individual model	88.3	61.0	59.1	92.5
	Greedy soup	88.3 (+0.0)	61.7 (+0.7)	59.1 (+0.0)	93.0 (+0.5)
T5 [215]	Best individual model	91.8	78.3	58.8	94.6
	Greedy soup	92.4 (+0.6)	79.1 (+0.8)	60.2 (+0.4)	94.7 (+0.1)

averages (EMA) of the weights [249] computed with decay factors of 0.999 (low EMA) and 0.9999999 (high EMA). Whereas high EMA generally provides the best single-model accuracy, both greedy soup and greedy ensembling attain higher validation accuracy when applied to parameters with low EMA. We report the highest single model accuracy numbers obtained with either EMA decay value, but perform greedy soup and ensembling with models trained with EMA decay of 0.999. For each combination of training run and EMA decay rate, we evaluate accuracy on our held out validation set every 1000 steps. We use these accuracy values to pick the best checkpoint for ensembling, souping, and subsequent evaluation.

In Table 3.4, we report results on the ImageNet validation set and the five distribution shift datasets studied above as well as two relabeled ImageNet validation sets, ReaL [22] and multilabel [232]. Our greedy soup procedure selects 14 of the 58 models fine-tuned as part of our hyperparameter sweep, and this soup performs statistically significantly better than the best individually fine-tuned model selected based on our held out validation set on all datasets except for ObjectNet. Even when we give an unfair advantage to individually fine-tuned models by selecting them based on their performance on each test set (denoted "oracle" in Table 3.4), the greedy soup, which was selected using only in-distribution data, remains superior on most datasets. Only on ReaL and ObjectNet does there exist an individual model that performs statistically significantly better than the soup, and the best model differs between those two datasets. Greedy ensembling performs similarly to the greedy soup in terms of ImageNet top-1 and multilabel accuracy, and slightly better on ReaL, but significantly worse on all distribution shift datasets except for ImageNet-V2. Thus, greedy soup can provide additional gains on top of standard hyperparameter tuning even in the extremely high accuracy regime.

Fine-tuning on text classification tasks

To test whether the gains obtained by model soups extend to domains beyond image classification, we conduct preliminary experiments with natural language processing (NLP). While more investigation is warranted to establish the applicability of model soups for NLP, we believe our experiments are a promising initial step. In particular, we fine-tune BERT [65] and T5 [215] models on four text classification tasks from the GLUE benchmark [259]: MRPC [69], RTE [51, 16, 93, 20], CoLA [264] and SST-2 [239], as in [68]. We use the standard metric for each dataset: average of accuracy and F_1 score for MRPC, accuracy for RTE,

Matthews correlation for CoLA [177] and accuracy for SST-2.

We fine-tune 32 models for each dataset with a random hyper-parameter search over learning rate, batch size, number of epochs and random seed. Table 3.5 reports the corresponding metric on the validation set for BERT-base uncased [64] and T5-base [215]. While the improvements are not as pronounced as in image classification, the greedy soup can improve performance over the best individual model in many cases.

3.5 Analytically comparing soups to ensembles

The goal of this section is to obtain complementary analytical insight into the effectiveness of model soups. For simplicity, we consider a soup consisting of only two models with parameters θ_0 and θ_1. For weighting parameter $\alpha \in [0, 1]$ we let $\theta_\alpha = (1-\alpha)\theta_0 + \alpha\theta_1$ denote the weight-averaged soup. We would like to understand when the soup error, $\mathrm{err}_\alpha := \mathbb{E}_{x,y} 1\{\mathrm{argmax}_i f_i(x;\theta_\alpha) \neq y\}$, would be lower that the best of both endpoints, $\min\{\mathrm{err}_0, \mathrm{err}_1\}$.

Note that just convexity of err_α in α does not by itself imply superiority of the soup to both endpoints, as the minimum of err_α over α may be obtained at the endpoints even when err_α is convex. To get further leverage on the problem, we compare the soup to the *logit-level ensemble* $f^{\mathrm{ens}}_\alpha(x) = (1-\alpha)f(x;\theta_0) + \alpha f(x;\theta_1)$. The rich literature on ensembles (see Sec. 2.7) tells us that the expected error of the ensemble, $\mathrm{err}^{\mathrm{ens}}_\alpha$, is often strictly below $\min\{\mathrm{err}_0, \mathrm{err}_1\}$ for neural networks. Therefore, whenever $\mathrm{err}_\alpha \approx \mathrm{err}^{\mathrm{ens}}_\alpha$ we expect the soup to outperform both endpoint models.

To analytically compare the soup and the ensemble, we replace the 0-1 loss with a differentiable surrogate. Specifically, we consider the cross-entropy loss $\ell(f, y) = \log\left(\sum_{y'} e^{f_{y'} - f_y}\right)$. We let $\mathcal{L}^{\mathrm{soup}}_\alpha = \mathbb{E}_{x,y}\ell(\beta f(x;\theta_\alpha), y)$ denote the β-calibrated expected loss of the soup, and similarly define $\mathcal{L}^{\mathrm{ens}}_\alpha = \mathbb{E}_{x,y}\ell(\beta f^{\mathrm{ens}}_\alpha(x), y)$ for the ensemble. We derive the following approximation for the loss difference:

$$\mathcal{L}^{\mathrm{soup}}_\alpha - \mathcal{L}^{\mathrm{ens}}_\alpha \approx \frac{\alpha(1-\alpha)}{2}\left(-\frac{\mathrm{d}^2}{\mathrm{d}\alpha^2}\mathcal{L}^{\mathrm{soup}}_\alpha + \beta^2 \mathbb{E}_x \mathrm{Var}_{Y\sim p_{\mathrm{sftmx}}(\beta f(x;\theta_\alpha))}\left[\Delta f_Y(x)\right]\right),$$

where $[p_{\mathrm{sftmx}}(f)]_i = e^{f_i}/\sum_j e^{f_j}$ is the standard "softmax" distribution and $\Delta f(x) = f(x;\theta_1) - f(x;\theta_0)$ is the difference between the endpoint logits. We obtain our approximation in the regime where the logits are not too far from linear.

The first term in the approximation is negatively proportional to the second derivative of the loss along the trajectory: when the approximation holds, convexity of the loss indeed favors the soup. However, the second term in the approximation does not follow from the "convex basin" intuition. This term always favors the ensemble, but is small in one of two cases: (a) the somewhat trivial case when the endpoint models are similar (so that Δf is small) and (b) when the soup produces confident predictions, implying that $p_{\text{sftmx}}(\beta f(x; \theta_\alpha))$ is close to a point mass and consequently the variance term is small.

To test our approximation, we evaluate it over of set of fine-tuned models with different learning rates, augmentation strategies, random seeds and α values. We set β to calibrate the soup model, and find that it improves the ability of our approximation to predict the soup/ensemble error difference.

When excluding the high learning rate of 10^{-4} (center and right panels),[3] we see that the approximation is strongly correlated with both the true difference in loss as well as the difference in error, and the approximation and true loss difference generally agree in sign.

3.6 Scope and limitations

While this work has so far demonstrated that averaging many fine-tuned models is a useful technique for improving accuracy, this section explores two limitations of the approach. The first is the applicability of model soups, and the second is the failure of model soups to substantially improve calibration.

Applicability. So far our experiments have mainly explored models pre-trained on large, heterogeneous datasets. We also explore model soups for an ImageNet-22k pre-trained model. While the greedy soup still provides improvements on ImageNet, these improvements are less substantial compared to those observed when fine-tuning CLIP and ALIGN.

Calibration. While ensembles improve model calibration [103, 225], model soups do not have the same effect. As hyperparameters can also have an effect on calibration, we consider the ensemble and soup of 20 models which are identical other than random seed.

[3] Fine-tuned models with learning rate 10^{-4} are far in weight space from the initial model and are often rejected when forming greedy soups. Therefore, we do not expect our approximation to be tight for these learning rates.

3.7 Related work

Averaging model weights. Averaging the weights of models is a popular approach in convex optimization and deep learning. Most applications study models along the same optimization trajectory, e.g. [227, 205, 249, 123, 298, 130, 129]. By contrast, Nagarajan and Kolter [190], Frankle et al. [82], Neyshabur et al. [192], Von Oswald et al. [258] and Matena and Raffel [176] weight-average models which share an initialization but are optimized independently. Nagarajan and Kolter [190] observed that models trained on MNIST [152] from the same random initialization are connected in weight space by a linear path of high accuracy. Frankle et al. [82] find that, when training a pair of models from scratch on harder datasets such as ImageNet with the same hyperparameter configuration and initialization but different data order, interpolating weights achieves no better than random accuracy. However, Frankle et al. [82] showed that when the two models share a portion of their optimization trajectory, accuracy does not drop when they are averaged. Analogously, Neyshabur et al. [192] demonstrate that when two models are fine-tuned with the same pre-trained initialization, the interpolated model attains at least the accuracy of the endpoints. Unlike Nagarajan and Kolter [190], Frankle et al. [82], Neyshabur et al. [192] we consider averaging many models with varied hyperparameter configurations.

In the late phases of training, Von Oswald et al. [258] make copies of a subset of the neural network parameters (e.g, the batch norm weights, the classification layer, etc.). These parameters are then optimized independently and subsequently averaged. In contrast to Von Oswald et al. [258], a) we average across independent runs with hyperparemter diversity, b) we modify all weights in the network, and c) we consider the transfer setting. Matena and Raffel [176] merge models with the same pre-trained initialization that are fine-tuned on different text classification tasks. They also propose Fisher information as an alternative technique for model merging. Wortsman et al. [272] average zero-shot and fine-tuned models, finding improvements in- and out-of-distribution. In contrast to Wortsman et al. [272], we average models across many independent runs which provides more substantial improvements.

Stochastic Weight Averaging (SWA) [123], which averages weights along a single optimization trajectory, is also motivated by the relation between ensembling model outputs and averaging model weights. In contrast, the averaging we propose is across independent runs. Moreover, while their analysis relates the averaged network outputs (i.e., the logit ensemble) to the output of the a network with the averaged weights,

our analysis (Section 3.5) goes a step further and relates the classification losses associated with these two vectors.

Pre-training and fine-tuning. In computer vision and natural language processing, the best performing models are often pre-trained on a large dataset before being fine-tuned on data from the target task [70, 285, 233, 97, 175, 142, 281, 139, 26]. This paradigm is also referred to as transfer learning. Recently, image-text pre-training has become increasingly popular in computer vision as a pre-training task [210, 125, 188, 202, 287]. Recent work has explored alternative strategies for adapting these models to specific target tasks [303, 89, 299], for instance via a lightweight residual feature adapter. In contrast, our work explores standard end-to-end fine-tuned models. Other work has attempted to improve transfer learning by regularizing models toward their initialization [278], choosing layers to tune on a per-example basis [104], reinitializing layers over the course of training [162], or using multiple pretrained models with data-dependent gating [237].

Ensembles. Combining the outputs of many models is a foundational technique for improving the accuracy and robustness of machine learning models [66, 18, 31, 85, 151, 84]. Ovadia et al. [198] show that ensembles exhibit high accuracy under distribution shift. Mustafa et al. [189] propose a method for identifying subsets of pre-trained models for fine-tuning and later ensembling them, finding strong in-distribution accuracy and robustness to distribution shift. Gontijo-Lopes et al. [101] conduct a large-scale study of ensembles, finding that higher divergence in training methodology leads to uncorrelated errors and better ensemble accuracy. Finally, previous work has explored building ensembles of models produced by hyperparameter searches [238, 181, 228], including greedy selection strategies [34, 35, 156, 267]. Importantly, ensembles require a separate inference pass through each model, which increases computational costs. When the number of models is large, this can be prohibitively expensive. Unlike ensembles, model soups require no extra compute at inference time.

3.8 Conclusion

Our results challenge the conventional procedure of selecting the best model on the held-out validation set when fine-tuning. With no extra compute during inference, we are often able to produce a better model by averaging the weights of multiple fine-tuned solutions.

Chapter 4

Patching open-vocabulary models by interpolating weights

4.1 Overview

Open-vocabulary models like CLIP achieve high accuracy across many image classification tasks. However, there are still settings where their zero-shot performance is far from optimal. We study *model patching*, where the goal is to improve accuracy on specific tasks without degrading accuracy on tasks where performance is already adequate. Towards this goal, we introduce PAINT, a patching method that uses interpolations between the weights of a model before fine-tuning and the weights after fine-tuning on a task to be patched. On nine tasks where zero-shot CLIP performs poorly, PAINT increases accuracy by 15 to 60 percentage points while preserving accuracy on ImageNet within one percentage point of the zero-shot model. PAINT also allows a single model to be patched on multiple tasks and improves with model scale. Furthermore, we identify cases of *broad transfer*, where patching on one task increases accuracy on other tasks even when the tasks have disjoint classes. Finally, we investigate applications beyond common benchmarks such as counting or reducing the impact of typographic attacks on CLIP. Our findings demonstrate that it is possible to expand the set of tasks on which open-vocabulary models achieve high accuracy without re-training them from scratch.

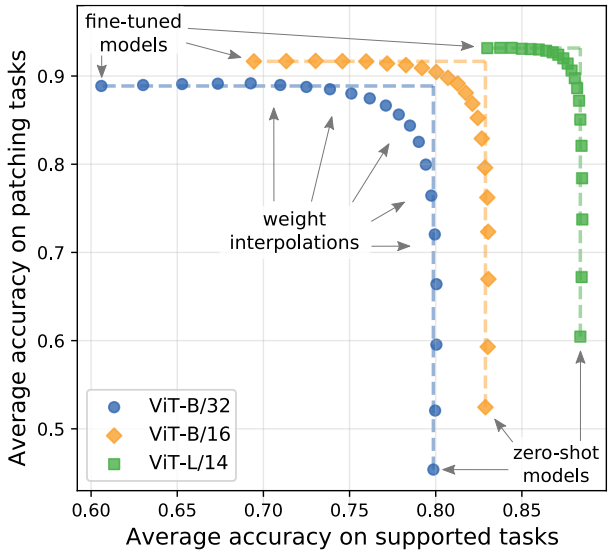

Figure 4.1: Patching open-vocabulary models by linearly interpolating weights. We wish to improve accuracy on tasks where a model performs poorly (*patching tasks*), without degrading performance on tasks where accuracy is already adequate (*supported tasks*). When interpolating weights of fine-tuned models and zero-shot (unpatched) models, there are intermediate solutions where accuracy improves on the patching task without reducing accuracy on supported tasks. Results are shown for CLIP models [210], averaged over nine patching tasks (Stanford Cars, DTD, EuroSAT, GTSRB, KITTI distance, MNIST, RESISC45, SUN397 and SVHN [143, 46, 109, 241, 91, 152, 40, 47, 276, 191]) and five supported tasks (ImageNet, CIFAR-10, CIFAR-100, STL-10 and Food101 [59, 144, 47, 29]). We apply PAINT separately on each patching task and average results across experiments. The dashed lines illustrate vertical movement from the unpatched models and horizontal movement from the fine-tuned models.

4.2 Introduction

Open-vocabulary models are characterized by their ability to perform any image classification task based on text descriptions of the classes [201]. Thanks to advances in large-scale pre-training, recent examples of open-vocabulary models such as CLIP and BASIC have reached parity with or surpassed important task-specific baselines, even when the open-vocabulary models are not fine-tuned on task-specific data (i.e., in a zero-shot setting) [210, 125, 201, 294, 4, 287]. For instance, the largest CLIP model from Radford et al. [210] used in a zero-shot setting matches the ImageNet accuracy of a ResNet-50 trained on 1.2 million ImageNet images [59, 107].

Nevertheless, current open-vocabulary models still face challenges. The same CLIP model that matches a ResNet-50 on ImageNet has lower MNIST accuracy than simple logistic regression in pixel space [210]. Moreover, even when zero-shot models achieve good performance, they are usually still worse than models

trained or fine-tuned on specific downstream tasks.

To address these issues, several authors have proposed methods for adapting zero-shot models to a task of interest using labeled data [272, 304, 89, 299, 147, 248]. A common practice is to fine-tune the zero-shot model on the task of interest [272, 201]. However, fine-tuned models can suffer from catastrophic forgetting [179, 251, 83, 137], performing poorly on tasks where the zero-shot model initially performed well [6, 272, 201]. Additionally, fine-tuning typically produces a task-specific classification head, sacrificing the flexible text-based API that makes open-vocabulary models so appealing. Whereas an open-vocabulary model can perform any classification task in a zero-shot fashion, a fine-tuned model with a task-specific head can only process the specific task that it was fine-tuned on. This specialization can prevent knowledge obtained by fine-tuning on one task from transferring to other related tasks with different classes.

Another approach to adapting zero-shot models would be to add data from the downstream task to the pre-training dataset and train a new open-vocabulary model from scratch. The resulting model could still perform any classification task, and zero-shot performance may improve on related tasks. However, training large image-text models from scratch can require hundreds of thousands of GPU hours [210, 201, 287], which makes this approach practically infeasible in most settings.

In this paper, we study *patching* open-vocabulary models, where the goal is to increase accuracy on new target tasks while maintaining the flexibility of the model and its accuracy on other tasks.[1] Patching aims to combine the benefits of fine-tuning and re-training from scratch: improved performance on the task of interest, maintaining the flexibility of an open vocabulary, transfer between tasks, and fast adaptation time. Motivated by these goals, we extend existing fine-tuning techniques [272] to open-vocabulary settings, where the class space is not fixed. We introduce Patching with Interpolation (PAINT), a simple, two-step procedure for patching models: first, fine-tune the model on the patching task without introducing any task-specific parameters; then, linearly interpolate between the weights of the model before and after fine-tuning. Linearly interpolating neural network weights [190, 82, 192] has been previously used to improve accuracy on a single task [123, 273] or robustness to distribution shift [272]. Indeed, averaging network weights has been explored in continual learning contexts, although for closed-vocabulary models [155].

With PAINT, accuracy can improve on new tasks without degrading accuracy on unrelated tasks, as

[1]The term *patching* is borrowed from software development terminology, drawing inspiration from recent work which conceptualizes developing machine learning models like open-source software [213, 224, 176, 247].

illustrated in Figure 4.1. For instance, applying PAINT to a CLIP ViT-L/14 [210] independently on nine image classification tasks [143, 46, 109, 241, 91, 152, 40, 276, 191] improves accuracy by 15 to 60 percentage points compared to the unpatched model, while accuracy on ImageNet [59] decreases by less than one percentage point. We also observe a promising trend: patching becomes more effective with model scale (Section 4.5.1).

Beyond single tasks, we show that models can be patched on multiple tasks (Section 4.6). When patching on nine image classification tasks simultaneously, a single CLIP ViT-L/14 model is competitive with using one specialized model for each task—the average accuracy difference is less than 0.5 percentage points.

Moreover, PAINT enables *broad transfer* (Section 4.7): accuracy on related tasks can increase, even when the class space changes. For instance, we partition EuroSAT [109], a satellite image dataset, into two halves with disjoint labels. Patching a ViT-L/14 model on the first half improves accuracy on the second half by 7.3 percentage points, even though the classes are unseen during patching.

Finally, we investigate PAINT on case studies including typographic attacks [99], counting [127], and visual question answering [8] (Section 4.8). For instance, applying PAINT using synthetic typographic attacks leads to a model that is less susceptible to typographic attacks in the real world, improving its accuracy by 41 percentage points.

In summary:

- Even the best pre-trained models are not perfect. We introduce PAINT, a method designed to improve accuracy on new tasks without harming accuracy elsewhere.
- PAINT incurs no extra computational cost compared to standard fine-tuning, neither during fine-tuning itself nor at inference time.
- PAINT can also be applied with multiple tasks, providing a single model that is competitive with many specialized models.
- Applying PAINT with one task can improve accuracy on a related task, even when they do not share the same classes.
- PAINT improves with model scale, indicating a promising trend for future models.

4.3 Patching with interpolation (PAINT)

This section details our method for patching models on a single and multiple tasks.

Patching on a single task. Given an open-vocabulary model with weights θ_{zs} and a patching task $\mathcal{D}_{\text{patch}}$, our goal is to produce a new model θ_{patch} which achieves high accuracy on $\mathcal{D}_{\text{patch}}$ without decreasing model performance on tasks where accuracy is already acceptable. We let $\mathcal{D}_{\text{supp}}$ denote a representative supported task where model performance is adequate, and later show that the method is stable under different choices of $\mathcal{D}_{\text{supp}}$ (Section 4.5.2). The two-step procedure we explore for producing θ_{patch} is given below.

Step 1. Fine-tune θ_{zs} on training data from $\mathcal{D}_{\text{patch}}$ to produce a model with weights θ_{ft}.

Step 2. For mixing coefficient $\alpha \in [0, 1]$, linearly interpolate between θ_{zs} and θ_{ft} to produce $\theta_{\text{patch}} = (1 - \alpha) \cdot \theta_{\text{zs}} + \alpha \cdot \theta_{\text{ft}}$. The mixing coefficient is determined via held-out validation sets for $\mathcal{D}_{\text{supp}}$ and $\mathcal{D}_{\text{patch}}$. We refer to the resulting model as θ_{patch}.

In our experiments, we do not introduce any additional task-specific parameters when fine-tuning, as discussed in Section 4.4.

Patching on a multiple tasks. In practice, we often want to improve model accuracy on multiple patching tasks $\mathcal{D}^{(1)}_{\text{patch}}, ..., \mathcal{D}^{(k)}_{\text{patch}}$, which can be accomplished with straightforward modifications to the procedure above. We explore three alternatives and examine their relative trade-offs in Section 4.6:

- *Joint patching*, where we merge all the patching tasks $\mathcal{D}^{(i)}_{\text{patch}}$ into a single task $\mathcal{D}_{\text{patch}}$ before running the patching procedure;
- *Sequential patching*, where we iteratively repeat the patching procedure above on each new task $\mathcal{D}^{(i)}_{\text{patch}}$ and let $\theta_{\text{zs}} \leftarrow \theta_{\text{patch}}$ after each completed iteration;
- *Parallel patching*, where we apply the first step on each task in parallel to produce fine-tuned models with weights $\theta^{(1)}_{\text{ft}}, ..., \theta^{(k)}_{\text{ft}}$. Then, we search for mixing coefficients α_i to produce $\theta_{\text{patch}} = (1 - \sum_{i=1}^{k} \alpha_i) \cdot \theta_{\text{zs}} + \sum_{i=1}^{k} \alpha_i \cdot \theta^{(i)}_{\text{ft}}$.

For joint and parallel patching we assume access to held-out validation sets for all tasks, while in sequential patching we only assume access to held-out validation sets from the tasks seen so far. Unless mentioned otherwise, we pick the mixing coefficient α that optimizes average accuracy on the held-out validation sets from the supported and patching tasks.

4.4 Experimental setup

Tasks. We consider a diverse set of image classification tasks from Radford et al. [210]. In most experiments, we use ImageNet [59] as a representative supported task, although we explore other supported tasks in Section 4.5.2. We categorize tasks into patching tasks or supported tasks based on the accuracy difference between the zero-shot model and a model specialized to the task. A large accuracy difference indicates that the task is a relevant target for patching because the zero-shot model is still far from optimal. Specifically, we consider a subset tasks from Radford et al. [210], categorizing tasks where the linear probes outperform the zero-shot model by over 10 percentage points as patching tasks: Cars [143], DTD [46], EuroSAT [109], GTSRB [241], KITTI [91], MNIST [152], RESISC45 [40], SUN397 [276], and SVHN [191]. We use the remaining tasks as supported tasks: CIFAR10 [144], CIFAR100 [144], Food101 [29], ImageNet [59], and STL10 [47]. We investigate additional patching tasks as case studies in Section 4.8 and provide further details in the Appendix.

Models. We primarily use CLIP [210] pre-trained vision transformer (ViT) models [71]. Unless otherwise mentioned our experiments are with the ViT-L/14 model, while Section 4.5.2 studies ResNets [107].

Fine-tuning on patching tasks. Unless otherwise mentioned, we fine-tune with a batch size of 128 for 2000 iterations using learning rate 1e-5 with 200 warm-up steps with a cosine annealing learning rate schedule and the AdamW optimizer [169, 200] (weight decay 0.1). When fine-tuning, we use the frozen final classification layer output by CLIP's text tower so that we do not introduce additional learnable parameters. This design decision keeps the model open-vocabulary and does not harm accuracy, as discussed in the Appendix.

Evaluation. We use accuracy as the evaluation metric unless otherwise stated. We refer to the average of the mean accuracy on the patching tasks and the mean accuracy on the supported tasks as *combined accuracy*.[2]

[2]In other words, $(\mathbb{E}_{\mathcal{D}_{\text{supp}}}[\mathsf{Acc}(\theta, \mathcal{D}_{\text{supp}})] + \mathbb{E}_{\mathcal{D}_{\text{patch}}}[\mathsf{Acc}(\theta, \mathcal{D}_{\text{patch}})])/2$, where $\mathsf{Acc}(\theta, \mathcal{D}_{\text{supp}})$ and $\mathsf{Acc}(\theta, \mathcal{D}_{\text{patch}})$ are accuracies on supported tasks and patching tasks, respectively.

4.5 Patching models on a single new task

As shown in Figure 4.1, when patching a model on a single task, we interpolate the weights of the zero-shot and fine-tuned model, producing a model that achieves high accuracy on both the patching task and the supported task. On the nine tasks, PAINT improves the accuracy of ViT-L/14 by 15 to 60 percentage points, while accuracy on ImageNet decreases by less than one percentage point. PAINT also allows practitioners to control the accuracy trade-off on the patching and supported tasks without re-training a new model, by varying the mixing coefficient α.

4.5.1 The effect of scale

We consistently observe that PAINT is more effective for larger models. Our findings are aligned with those of Ramasesh et al. [217], who observed that larger models are less susceptible to catastrophic forgetting. This section formalizes and provides insights for these observations.

Measuring the effectiveness of patching. We measure the effectiveness of patching via the accuracy difference between the single patched model and two specialized models with the same architecture and initialization. For both the supported task and patching task, we take specialized models that maximize performance on the task, considering the set of all interpolations between the zero-shot and fine-tuned models. We refer to this measure as *accuracy distance to optimal*. Formally, accuracy distance to optimal is given by

$$\frac{1}{2}\left[\max_{\alpha} \mathsf{Acc}(\theta_\alpha, \mathcal{D}_{\text{supp}}) + \max_{\alpha} \mathsf{Acc}(\theta_\alpha, \mathcal{D}_{\text{patch}})\right] - \frac{1}{2}\max_{\alpha}\left[\mathsf{Acc}(\theta_\alpha, \mathcal{D}_{\text{supp}}) + \mathsf{Acc}(\theta_\alpha, \mathcal{D}_{\text{patch}})\right], \tag{4.1}$$

where $\mathrm{Acc}(\theta, \mathcal{D})$ represents the accuracy of model θ on task $\mathcal{D}$. In Figure 4.2 (left), we show that accuracy distance to optimal decreases with scale, indicating that patching becomes more effective for larger models.

Model similarity. Fine-tuning modifies overparameterized models less, which provides insights on why larger models are easier to patch: less movement is required to fit new data. We demonstrate this by evaluating representational similarity using Centered Kernel Alignment (CKA) [141] (see Appendix for details). As shown in Figure 4.2 (center), the representations of the unpatched and fine-tuned models be-

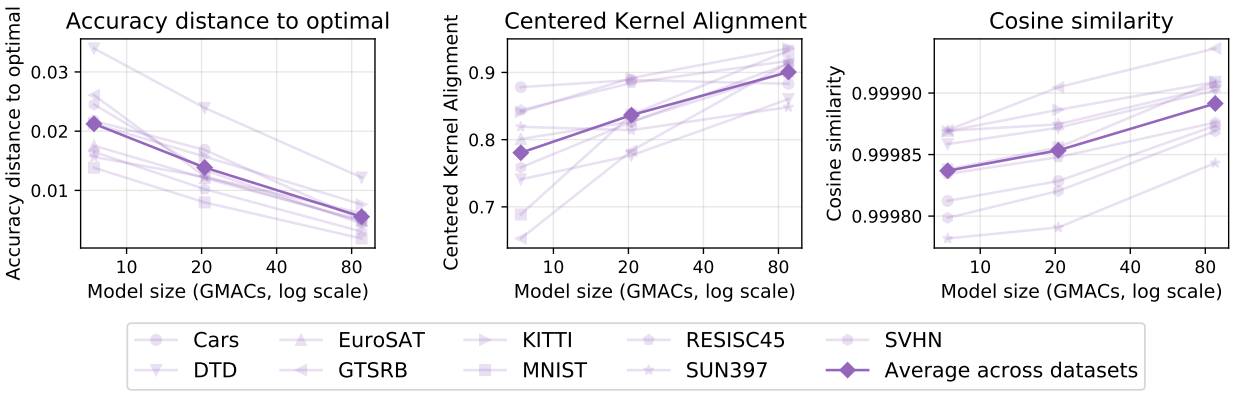

Figure 4.2: Larger models are easier to patch (left). For larger models, the unpatched and fine-tuned model are more similar with respect to their representations (center) and weights (right). Model scale is measured in Giga Multiply-Accumulate operations (GMACs).

come more similar as models grow larger, indicated by larger CKA values. Moreover, Figure 4.2 (right) shows that the cosine similarity between the weights of the unpatched and fine-tuned models, $\cos(\theta_{\text{zs}}, \theta_{\text{ft}}) = \langle \theta_{\text{zs}}, \theta_{\text{ft}} \rangle / (||\theta_{\text{zs}}|| \, ||\theta_{\text{ft}}||)$, increases with scale.

4.5.2 Baselines and ablations

Baselines. There are many alternatives which enable a trade-off between accuracy on the supported and patching tasks. These methods include early stopping during fine-tuning, applying a regularization term which penalizes movement from initialization, or training with different hyperparameters including a smaller learning rate. Unlike interpolation, these methods do not enable navigating the accuracy trade-off without fine-tuning the model again many times. Moreover, Figure 4.3 demonstrates that the accuracy trade-off frontier for early stopping, regularization, or varying hyperparameters can be recovered by interpolating weights with different mixing coefficients.

Additional supported tasks. In Figure 4.1, we use ImageNet as a representative supported task. This section demonstrates that PAINT is stable under different choices of the supported task. Instead of ImageNet, we use CIFAR10, CIFAR100, Food101 and STL10. Figure 4.4 displays representative results, where performance is averaged over the nine patching tasks. We observe consistent results across supported tasks, and that the optimal mixing coefficients are stable across different choices of supported tasks (Figure 4.4,

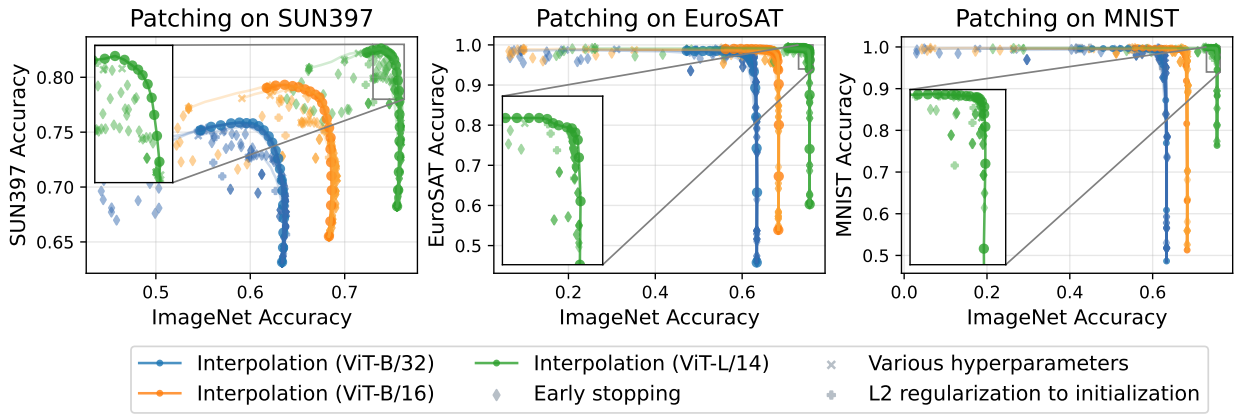

Figure 4.3: The frontier of accuracy trade-offs can be recovered by linearly interpolating weights. Interpolating the unpatched and fine-tuned models recovers the accuracy trade-off of early stopping, regularization towards the initialization, and changes in hyperparameters. Additional details and comparisons can be found in the Appendix.

right).

4.6 Patching models on multiple tasks

This section details experimental results for patching on multiple datasets. Recall from Section 4.3 that there are various strategies for extending PAINT to multiple datasets, which we briefly revisit. For *joint* patching we merge all the datasets into a single fine-tuning task and apply our patching procedure as before. For *sequential* patching we iteratively perform our procedure once per task, using the patched model at each step as the initialization for the next step.[3] We also explore *parallel* patching, for which we have an unpatched model θ_{zs} and independently fine-tune on each of the tasks in parallel. We then search for mixing coefficients to combine the resulting models. For tasks $1, ..., k$, let $\theta_{\text{ft}}^{(1)}, ..., \theta_{\text{ft}}^{(k)}$ denote the fine-tuned models for each task. Since it is impractical to exhaustively search over each α_i, we instead search over a one-dimensional scalar $\alpha \in [0, 1]$, which interpolates between θ_{zs} and the average of all fine-tuned solutions $\frac{1}{k}\sum_{i=1}^{k}\theta_{\text{ft}}^{(i)}$.[4]

These methods have various trade-offs and may be applicable for different scenarios. Joint patching is only possible when data from all tasks you wish to patch is available. On the other hand, sequential

[3] The results are averaged over three random seeds that control the order in which tasks are seen.

[4] We also explored adaptive black-box optimization algorithms to choose the mixing coefficients α_i [219], but observed little improvement (0.3 to 0.4 percentage points on average).

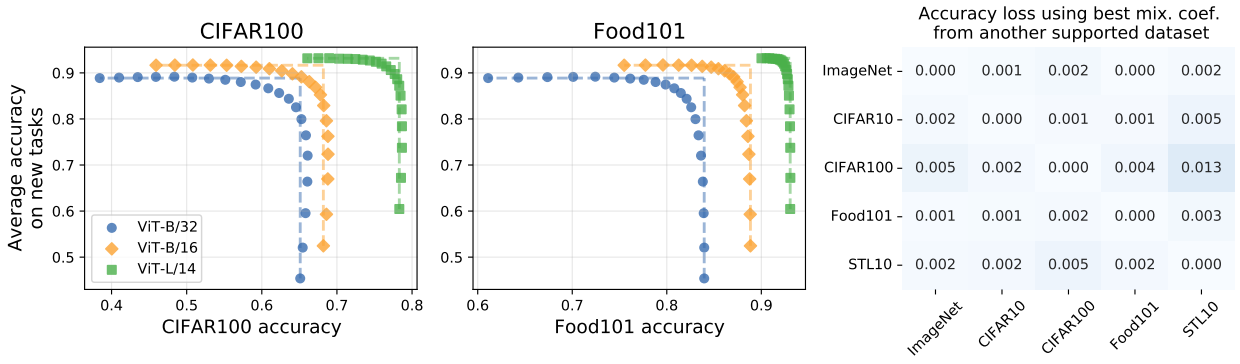

Figure 4.4: Results are consistent across supported tasks. For multiple supported tasks, we observe similar accuracy improvements on patching tasks, without substantially decreasing supported task accuracy. Moreover, choosing the mixing coefficients using a different supported task does not substantially decrease combined accuracy on patching and supported tasks (right).

patching is appropriate when the tasks are observed one after another. Finally, parallel patching can leverage distributed hardware.

Figure 4.5 displays experimental results when patching on all nine tasks from Section 4.5. We observe that joint patching is the best-performing method on average. This is perhaps unsurprising since joint patching has simultaneous access to all patching datasets, unlike other patching strategies. Nevertheless, it is still interesting that for ViT-L/14, joint patching yields a *single* model with only 0.5 percentage points worse combined accuracy than using *multiple* specialized models.[5] Joint patching also achieves a 15.8 percentage points improvement over the unpatched model. Moreover, patching a ViT-B/32 model with the joint strategy achieves a combined accuracy 6.1 percentage points higher than a ViT-L/14 unpatched model, which requires 12x more GMACs.

The accuracy of sequential patching approaches that of joint patching, especially for larger models. Note that, unlike in joint patching, forgetting can compound since the patching procedure is applied multiple times in sequence. In sequential patching, weight interpolations do not completely eradicate forgetting, but greatly mitigate it. This is most noticeable for smaller models: sequentially fine-tuning a ViT-B/32 without interpolation *reduces* the combined accuracy by 4.6 percentage points compared to the unpatched model. This is compared to a combined accuracy *increase* of 11 percentage points when using sequential patching. Additional results, including experiments on SplitCIFAR [220], can be found in the Appendix.

Finally, parallel patching underperforms other patching strategies. Like sequential patching, parallel

[5]Recall from Section 4.4 that combined accuracy weight patching and supported tasks equally.

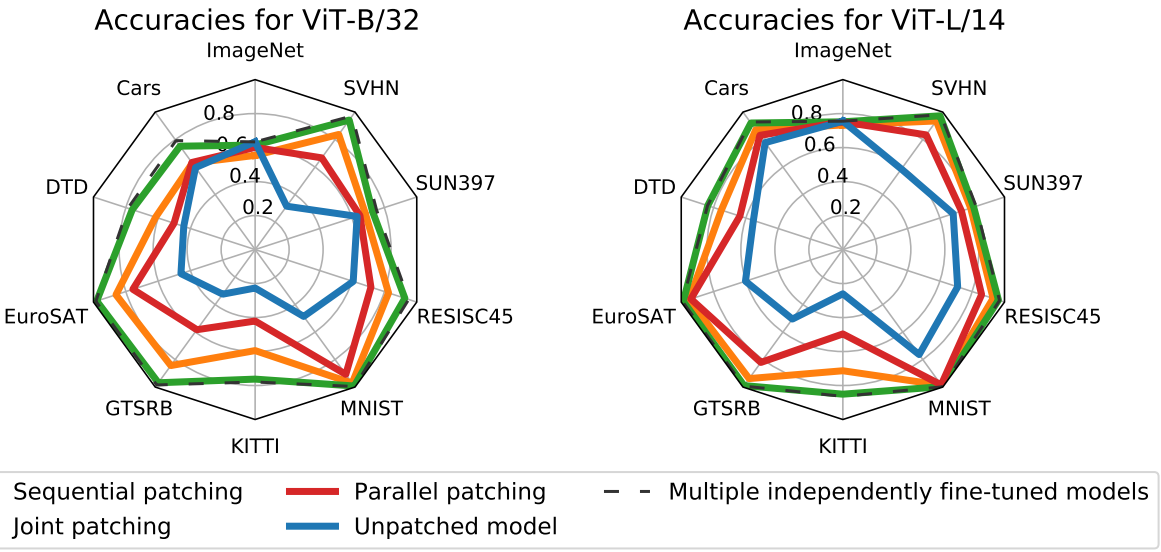

Figure 4.5: Contrasting various strategies for patching on multiple tasks. On all experiments, ImageNet is used as the supported task while the other nine datasets are used for patching. When data from all patching tasks is available, joint patching yields a single model that is competitive with using ten different specialized models. Weight interpolations greatly mitigate catastrophic forgetting on the sequential case, but do not completely eradicate it. Finally, parallel patching underperforms other patching strategies, but still provides improvements over the unpatched model.

	Cars	DTD	EuroSAT	GTSRB	KITTI	MNIST	RESISC45	SUN397	SVHN
Unpatched accuracy	86.2	64.9	79.9	51.7	43.4	82.6	73.4	76.9	72.8
Patched accuracy	87.0 **(+0.8)**	66.1 **(+1.2)**	87.2 **(+7.3)**	71.1 **(+19.4)**	60.4 **(+17.0)**	91.3 **(+8.7)**	74.2 **(+0.8)**	79.3 **(+2.4)**	88.9 **(+16.1)**

Table 4.1: PAINT can generalize to unseen classes. We randomly partition each dataset into tasks A and B with disjoint class spaces of roughly equal size. This table reports how patching on task A affects accuracy on task B for the ViT-L/14 model. In all cases, accuracy on task B improves when patching on task A even though the classes are *unseen* during patching.

patching is in the challenging setting where data from all patching tasks is not available simultaneously. Moreover, unlike in joint or sequential patching, no model is optimized on data from all patching tasks. Using a black box optimization algorithm for finding the mixing coefficients did not yield large improvements over using the same mixing coefficient for all models. However, it is possible that more sophisticated search methods could yield better results. In the Appendix, we present additional experiments for a subset of the tasks where exhaustively searching the space of mixing coefficients is tractable, finding headroom for improvement in most cases.

Task A	MNIST	SVHN	EuroSAT	RESISC45	MNIST	FashionMNIST	GTSRB	MTSD
Task B	SVHN	MNIST	RESISC45	EuroSAT	FashionMNIST	MNIST	MTSD	GTSRB
Unpatched accuracy	58.6	76.4	71.0	60.2	67.7	76.4	19.3	50.6
Patched accuracy	68.9 **(+10.3)**	93.2 **(+16.8)**	69.7 **(-1.3)**	70.4 **(+10.2)**	70.8 **(+3.1)**	77.5 **(+1.1)**	30.8 **(+11.5)**	69.8 **(+19.2)**

Table 4.2: Patching on task A can improve accuracy on a related task B. For a pair of tasks A and B, we report accuracy of the ViT-L/14 on task B, after patching on task A, finding improvements on seven out of eight cases.

4.7 Broad transfer

An alternative to our patching approach is to introduce parameters which are specific to each new task. By contrast, PAINT always maintains a single model. This section describes an additional advantage of the single model approach: patching the model on task A can improve accuracy on task B, even when task A and B do not share the same classes. We refer to this phenomenon as *broad transfer*. Note that we are able to study this phenomenon because the single patched model remains open-vocabulary throughout the patching procedure. This is a key advantage of PAINT compared to maintaining a collection of task-specific models.

We now describe two experiments to measure the effects on a task B when patching the model on a task A. First, we explore broad transfer by randomly partitioning datasets into disjoint sets with no class overlap. For a dataset $\mathcal{D}$ we partition the class space $\mathcal{Y}$ into two disjoint sets of roughly equal size $\mathcal{Y}_A$ and $\mathcal{Y}_B$. We build task A with the examples $(x, y) \in \mathcal{D}$ where y belongs to $\mathcal{Y}_A$, and task B with examples (x, y) where y belongs to $\mathcal{Y}_B$. Table 4.1 shows how patching a model on task A affects the accuracy on task B for nine datasets $\mathcal{D}$. The accuracy improvements on task B range from 0.8 to 19.4 percentage points, even though the classes from task B are not seen during patching.

To further understand transfer, we consider additional task pairs A and B, which are now different datasets. While some pairs A, B share classes, there are still instances of broad transfer. Concretely, Table 4.2 examines i) MNIST and SVHN, two digit recognition tasks with shared classes; ii) EuroSAT and RESISC45, two satellite imagery recognition tasks where there are unshared classes but some overlap; iii) GTSRB and MTSD [76], two traffic sign recognition datasets where there are unshared classes but some overlap; and iv) MNIST and FashionMNIST [275], which do not share any classes but appear visually

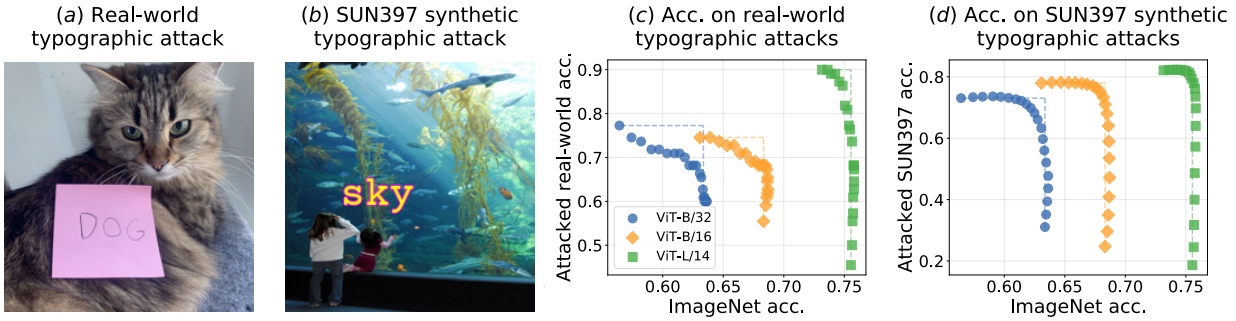

Figure 4.6: Guarding against real-world typographic attacks by patching on synthetic data. *(a)* A sample from our real-world typographic attacks test set. A CLIP ViT-L/14 is "tricked" into classifying this image as a dog instead of a cat. *(b)* Sample of synthetic typographic attack data. *(c)* Performance on real-world data with unseen classes after patching on *only* synthetic typographic attacks (curves produced by interpolating between the unpatched and fine-tuned model). *(d)* Analogous curves for the test set of the synthetic data used for patching.

similar. In seven out of eight experiments, patching on task A improves accuracy by 1.1 to 19.2 percentage points on task B. The exception is when A is EuroSAT and B is RESISC45, where accuracy decreases by 1.3 percentage points.

In all experiments, when patching on task A we choose the mixing coefficient α by optimizing the held-out validation accuracy on task A and a supported task (in this experiment we use ImageNet). While it is possible for a method that introduces new parameters for each task to exhibit broad transfer to new data, this also requires knowing which parameters to apply for the new data. This is not necessary in the single model approach.

4.8 Case studies

We further examine the performance of PAINT in three additional settings, which highlight weaknesses of the zero-shot CLIP model and showcase broad transfer (Section 4.7).

Typographic attacks. Goh et al. [99] find that CLIP models are susceptible to *typographic attacks*, where text superimposed on an image leads to misclassification. For example, in Figure 4.6 *(a)*, the text on the pink note saying "dog" leads a CLIP to misclassify the image of a cat as a dog. To fix this vulnerability, we procedurally generate typographic attack data by adding text with incorrect class names to SUN397 [276],

as seen in Figure 4.6 *(b)*. We then collect a test set of 110 real world images by placing notes on objects and taking photos.[6] After applying PAINT using the synthetic data, we evaluate on the real-world images (Figure 4.6 *(c)*) and synthetic test set (Figure 4.6 *(d)*). We observe that while larger models are more susceptible to typographic attacks, they are also more amenable to patching. Furthermore, we see an example of broad transfer between the synthetic and real-world data: when patching ViT-L/14 on synthetic data, its accuracy on real-world typographic attacks improves 41 percentage points even though the real-world classes are unseen. The cost is a reduction of less than 1 percentage point on ImageNet.

Counting. Radford et al. [210] find that CLIP models struggle to count the number of objects in CLEVR [127]. Here, the task is to choose an integer between 3 and 10 for each image, corresponding to the number of visible objects. While a straightforward way to patch such a task is to fine-tune on it directly, we investigate if applying PAINT using a subset of the classes allows the patched model to generalize to other numbers. Specifically, we patch on images with 4, 5, 6, 8, or 9 objects. To evaluate broad transfer, we test on images with 3, 7, and 10 objects (7 for understanding interpolation and 3 and 10 for extrapolation). We find that PAINT improves accuracy from 59% to over 99% on unseen classes with less than half a percentage point decrease in ImageNet accuracy.

Visual question answering. As shown by Shen et al. [236], zero-shot CLIP models perform poorly on visual question answering [8]. Using CLIP for VQA typically involves additional parameters—for instance, Shen et al. [236] trains a transformer [257] on CLIP features. In contrast, our procedure for patching CLIP on VQA does not introduce new parameters. Following Shen et al. [236], we contrast images with a series of text prompts, where each prompt corresponds to an option in multiple-choice VQA, formed by both the question and a candidate answer using the following template: "Question: [question text] Answer: [answer text]". We evaluate on multiple-choice VQA v1 [8], where each question is associated with 18 candidate answers. Our results show that patching is effective for visual question answering: PAINT improves the accuracy of a ViT-L/14 model by 18 percentage points, while accuracy drops by less than one percentage point on ImageNet.

[6] Data available at https://github.com/mlfoundations/patching.

4.9 Limitations and conclusion

Limitations. When applying PAINT, accuracy on supported tasks can still decrease, especially for smaller models. This limitation is perhaps best reflected in the case of sequential patching: patched models underperform using multiple specialized models when many tasks are added sequentially. Using larger models and weight interpolations can alleviate this issue, but do not completely resolve it. Finally, better understanding on which datasets patching is more effective is an exciting direction for future research.

Conclusion. In this work, we explore several techniques for patching open-vocabulary models with the goal of improving accuracy on new tasks without decreasing accuracy elsewhere. PAINT is effective in several scenarios, ranging from classifying digits to defending against typographic attacks. PAINT becomes more effective with scale, and can be applied on multiple tasks sequentially or simultaneously. Our findings demonstrate that in many circumstances it is possible to expand the set of tasks on which models achieve high accuracy, without introducing new parameters, without re-training them from scratch, and without catastrophic forgetting.

Chapter 5

lo-fi: distributed fine-tuning without communication

5.1 Abstract

When fine-tuning large neural networks, it is common to use multiple nodes and to communicate gradients at each optimization step. By contrast, we investigate completely local fine-tuning, which we refer to as lo-fi. During lo-fi, each node fine-tunes independently without any communication. Then, the weights are averaged across nodes at the conclusion of fine-tuning. When fine-tuning DeiT-base and DeiT-large on ImageNet, this procedure matches accuracy in-distribution and improves accuracy under distribution shift compared to the baseline, which observes the same amount of data but communicates gradients at each step. We also observe that lo-fi matches the baseline's performance when fine-tuning OPT language models (up to 1.3B parameters) on Common Crawl. By removing the communication requirement, lo-fi reduces resource barriers for fine-tuning large models and enables fine-tuning in settings with prohibitive communication cost.

5.2 Introduction

Many of the best performing machine learning models today come from a two step procedure: First, *pre-train* on a large, heterogeneous dataset to learn a good representation. Next, *fine-tune* to adapt the model to

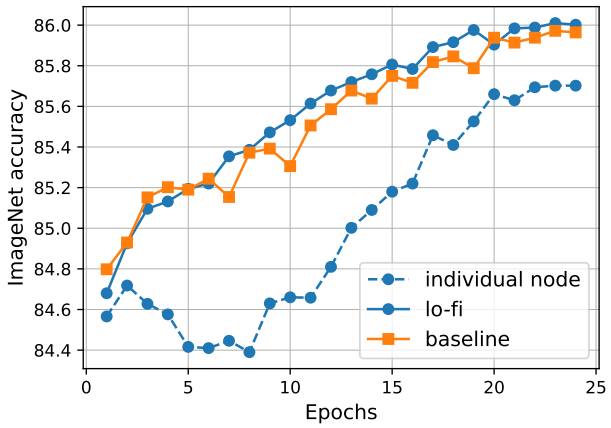

Figure 5.1: In standard multi-node distributed data-parallel fine-tuning, there is synchronization between nodes at each step of fine-tuning. With lo-fi (local fine-tuning), there is no communication between nodes throughout fine-tuning. As a result, each node k independently produces their own model θ^k. Then, lo-fi averages these models once for the final solution $\theta_{\text{lo-fi}} = \frac{1}{n}\sum_{k=1}^{n}\theta^k$. In this four-node fine-tuning run, we show (i) the average accuracy of the individual models θ^k, (ii) the accuracy of $\theta_{\text{lo-fi}}$ at the end of each fine-tuning epoch, and (iii) the accuracy of the baseline which communicates among nodes every step. In particular, we fine-tune the ImageNet-21k pre-trained DeiT-base model from DeiT-III [255] on ImageNet [59] using their code, which uses four nodes.

	IN	IN-V2	IN-R	Sketch	IN-A
baseline (DeiT-b)	85.96	76.65	62.66	46.86	57.15
lo-fi (DeiT-b)	86.00	76.84	63.25	48.37	58.43
baseline (DeiT-l)	87.12	78.18	69.87	54.41	68.97
lo-fi (DeiT-l)	87.10	78.25	70.14	54.95	69.53

Table 5.1: Comparing lo-fi (no communication during fine-tuning) to the baseline which communicates at each step when fine-tuning the ImageNet-21k pre-trained DeiT-base and DeiT-large model from DeiT-III [255] on ImageNet [59]. Both lo-fi and the baseline use the same number of iterations, which have been tuned for the baseline. Underlined numbers indicate significantly better accuracy according to McNemar's test with significance level 0.05. Lo-fi matches performance on ImageNet (IN), but can outperform the baseline on some distribution shifts. The shifts we consider are IN-V2 [222], IN-R [112], Sketch [260], and IN-A [113].

a task of interest [97, 285, 142, 139]. This paper operates within the second step of this procedure—fine-tuning—which is increasingly important with drastic improvements in pre-trained models, e.g., CLIP [210], GPT-3 [33], OPT [301], and PaLM [44]. Indeed, recent advances such as Minerva [157] or InstructGPT [197] have come from fine-tuning rather than training from scratch.

Most work developing learning methods still operates in the paradigm of training from scratch. Accordingly, both use similar algorithmic techniques despite important differences in the pre-training and fine-tuning regimes. In particular, one notable difference between pre-training and fine-tuning is that fine-tuned models appear to lie in a single low-error region [192]. Indeed, linearly interpolating the weights of fine-tuned models can have similar advantages as ensembling their predictions but without the added cost during inference [273]. By contrast, linearly interpolating the weights of two models trained from scratch will encounter a high error barrier [82, 90].

Recently, the model soups approach [273] leveraged this similarity between ensembling outputs and averaging weights. Given a hyperparameter sweep over fine-tuned models, they average the weights of multiple models instead of the conventional procedure of selecting one model and discarding the remainder. However, the model soups approach does not modify the fine-tuning procedure itself.

In this paper, we leverage the observation that fine-tuned models appear to lie in a single low error region to remove communication between nodes during distributed fine-tuning. In standard data-parallel multi-node fine-tuning, gradients between nodes are communicated at each step. This synchronization of updates keeps the models at each node identical to each other during fine-tuning. However, in certain settings communication costs during fine-tuning may be prohibitive, and we therefore ask whether they are necessary at all. With our method of local fine-tuning, which we refer to as *lo-fi*, we remove all communication between nodes during fine-tuning. The models on each node therefore drift apart throughout fine-tuning. Then, to arrive at the final solution at the end, we average the weights of the models produced by each node.

We note that these techniques are a natural extension of previous work: lo-fi is just a *model soup* [273] formed by splitting up a large fine-tuning job into multiple smaller jobs, each isolated to a node. Analogously, lo-fi is embarrassingly parallel training from *branch-train-merge* [160] applied in the setting where no domain specialization info is provided and so each expert is trained on IID data. However, we believe that the application of these techniques in this setting is of practical interest, especially if models continue

	IN	IN-V2	IN-R	Sketch	IN-A	epochs	stoch. depth	no extra cost	no comms
DeiT-base									
[255]	85.72	76.53	61.83	47.44	57.29	50	0.15	✓	
baseline	85.96	76.65	62.66	46.86	57.15	24	0.15	✓	
individual node	85.66	76.22	62.09	46.75	56.00	6	0.10	✓	✓
lo-fi	86.00	76.84	**63.25**	**48.37**	**58.43**	24	0.10	✓	✓
lo-fi ensemble	**86.08**	**76.91**	63.05	47.67	57.80	24	0.10		✓
DeiT-large									
[255]	86.97	78.47	69.70	54.35	68.57	50	0.40	✓	
baseline	87.12	78.18	69.87	54.41	68.97	12	0.30	✓	
individual node	86.76	78.00	69.41	54.57	67.59	3	0.25	✓	✓
lo-fi	87.10	78.25	**70.14**	**54.95**	**69.53**	12	0.25	✓	✓
lo-fi ensemble	**87.14**	**78.35**	70.00	54.62	69.20	12	0.25		✓

Table 5.2: Expanding the comparison between lo-fi and the baseline (Table 5.1) when fine-tuning the ImageNet-21k pre-trained models from DeiT-III [255] on ImageNet [59]. In this four node fine-tuning run, lo-fi removes communication between nodes so that each node produces an independent model. The weights of the models are then averaged at the end to produce the final solution. In this table we bold the highest number and evaluate the following models: i) paper, the fine-tuned models from the DeiT-III paper [255], ii) baseline, which is our improved fine-tuning baseline after hyperparameter turning which requires less epochs of training but achieves slightly higher accuracy than reported in the DeiT-III paper [255], iii) individual node, which is one of the individual node models that is produced by lo-fi, iv) lo-fi, which fine-tunes individual models on each node then averages their weights once at the end, and v) lo-fi ensemble which averages the outputs of the models produced by each node during lo-fi, and therefore requires more cost during inference. In addition to evaluating on ImageNet (IN), the task used for fine-tuning, we also evaluate on the distribution shifts ImageNet-V2 (IN-V2, [222]), ImageNet-R (IN-R, [112]), ImageNet-Sketch [260], and ImageNet-A (IN-A [113]). While more information is provided in Section 5.4.1, this table also displays some hyperparameter changes we made from the default DeiT-III fine-tuning script. Unlike [255], we fine-tune with LP-FT [148], and observe it is better for the baseline to use fewer fine-tuning epochs. Lo-fi observes the same amount of data as the tuned baseline and uses the same hyperparameters with the exception of slightly decreased regularization by lowering stoch. depth [118] drop probability by 0.05 (making the same change to the baseline decreased accuracy). Additional columns track whether the model incurs no additional cost during inference compared to a single model (denoted no extra cost), and also if there is no communication between nodes during fine-tuning (denoted no comms). Overall, lo-fi matches or outperforms the baseline without communication between nodes during fine-tuning.

to grow.

In computer vision we use the DeiT-III codebase [255] to fine-tune the ImageNet-21k pre-trained DeiT-base and DeiT-large models, which are four-node fine-tuning jobs by default. We observe (Figure 5.1, Table 5.1) that lo-fi matches the accuracy of DeiT-base and DeiT-large on ImageNet, the task used for fine-tuning, while outperforming the baseline on some distribution shifts. These improvements come after hyperparameter tuning the baseline to slightly exceed that in the DeiT-III paper while requiring fewer fine-tuning epochs. Moreover, lo-fi and the baseline observe the same amount of data. While overall similar results are observed when fine-tuning CLIP ViT-L [210] on ImageNet or tasks from WILDS [138], lo-fi often requires more iterations in this setting. Finally, we test lo-fi beyond computer vision by fine-tuning OPT-125M and OPT-1.3B [301] on Common Crawl, observing that lo-fi can match the baseline which communicates between nodes.

Overall, our work is a test of whether communication between nodes is required during fine-tuning. However, we also wanted to understand the advantages of removing this communication. Therefore, we benchmark the wall-clock overhead of communication on an AWS cluster with EFA. We use the models from the DeiT-III repository [255] in the context of image classification. In this setting and on the system used for this study, the advantages are overall less substantial than we initially expected, especially for large batch sizes. Notably, we observe that the trick of overlapping the communication and computation in the backwards pass [161], which is the default in PyTorch [200] as of v1.5, reduces the overhead of using multiple nodes from roughly 50% slow-down to under 10% for the large DeiT model. Finally, we discuss how lo-fi can help with faster job scheduling and addresses the straggler and jitter problem in distributed training, where different nodes might experience random slowdowns.

5.3 Methods

This section details the methods used in our experiments. We begin with the baseline of standard data-parallel training, and next outline our straightforward modification which i) removes communication between nodes then ii) averages the final models produced by each node.

Consider a neural network $f(x, \theta)$ where x is the input data and $\theta \in \mathbb{R}^d$ are the network parameters. Since we are fine-tuning, θ is initialized as the weights of a pre-trained model. Moreover, as is standard

in neural network training, the input data x is a batch rather than a single data point. Finally, let n denote the number of devices, b denote total batch size, and $\ell(\hat{y}, y)$ denote loss for the vector of predicted labels $\hat{y} = f(x, \theta)$ and a vector of ground-truth labels y.

With communication. The most straightforward and common approach for training with n devices is data-parallel. In this setting, each device has their own copy of the parameters θ. During fine-tuning, each batch x of size b is split into n disjoint sub-batches of size b/n. Each device i loads the sub-batch (x_i, y_i) and computes gradients $g_i = \nabla_\theta \ell(f(x_i, \theta), y_i)$. Then the gradients are synchronized across nodes with each node computing an averaged gradient $\bar{g} = \frac{1}{n} \sum_{i=1}^{n} g_i$. After synchronizing gradients, each device uses $\bar{g}$ to update θ. Since every device updates θ using an identical gradient $\bar{g}$, the parameters θ remain identical across devices.

lo-fi. With local-finetuning (lo-fi), we partition the n devices into K disjoint groups. In the majority of our experiments, each group is a single node containing 8 GPU devices. During fine-tuning we allow communication within each group, but not across groups. Each group k begins with parameters θ^k which are initially identical across devices, but drift apart throughout fine-tuning. Then, at the end of fine-tuning there is a single communication and the parameters from each group are averaged to produce a final solution $\theta = \frac{1}{K} \sum_{k=1}^{K} \theta^k$.

There are two possible implementations for lo-fi which we refer to as implementation A and B. Implementation A proceeds as before—each device i loads the sub-batch (x_i, y_i) and computes gradients g_i. There is then gradient synchronization only among devices belonging to the same group while devices from different groups apply different gradients. Data partitioning is accomplished without communication by coordinating random seeds, so long as each device knows its rank and the total number of devices. Our experiments primarily use Implementation A.

In Implementation B, each group is a completely independent run—no knowledge of total number of devices is required. Accordingly, within each group the global batch size is scaled by $1/K$ so that the per-device batch size is matched. Our image classification results use Implementation A while our language modelling results use Implementation B.

Our motivation for having one group per node and still allowing communication among the devices on

the node is that communication within a node is faster than communication across nodes.

5.4 Experiments

This section presents our experiments which test whether communication is required during fine-tuning. First we use the DeiT-III codebase [255] to fine-tune their pre-trained ImageNet-21k models on ImageNet, where we observe that lo-fi matches the baseline but without communication between nodes (Section 5.4.1). Next, we fine-tune CLIP [210] on ImageNet, WILDS-FMoW [138, 45] and WILDS-iWildCam [19] (Section 5.4.2). Finally, we show preliminary experiments applying lo-fi outside of computer vision (Section 5.4.3) and benchmark the associated speed-ups by removing communication (Section 5.4.4).

5.4.1 Fine-tuning DeiT-III on ImageNet

The aim of these experiments is to test whether communication between nodes is required when fine-tuning high accuracy models for image classification. To test this we begin by fine-tuning the DeiT-base and DeiT-large models from the DeiT-III paper [255] using their code. In particular, we fine-tune their ImageNet-21k models on ImageNet-1k [59] with and without lo-fi.

We chose the models from DeiT-III for a few reasons: (i) DeiT-III is representative of state-of-the-art settings as it uses many advanced techniques such as stochastic depth [118], CutMix [289], and the LAMB optimizer [286]. (ii) DeiT-III provides hyperparameter configurations which they used in their fine-tuning experiments. (iii) DeiT-III uses 4 nodes with 8 GPUs each when fine-tuning their pre-trained ImageNet-21k models on ImageNet. This provides an opportunity to test lo-fi in an equivalent setting where there is normally communication between nodes.

Main results. Our overall finding is that communication between nodes is not necessary in this setting—lo-fi matches the accuracy of the baseline while observing the same amount of data. These results are presented in Figure 5.1 and Tables 5.1 and 5.2. In these experiments, lo-fi uses 4 groups—each group corresponds to one node.

Figure 5.1 illustrates accuracy throughout training when fine-tuning DeiT-base with and without lo-fi. We also report the average accuracy of the models produced by the individual nodes. To make this plot we

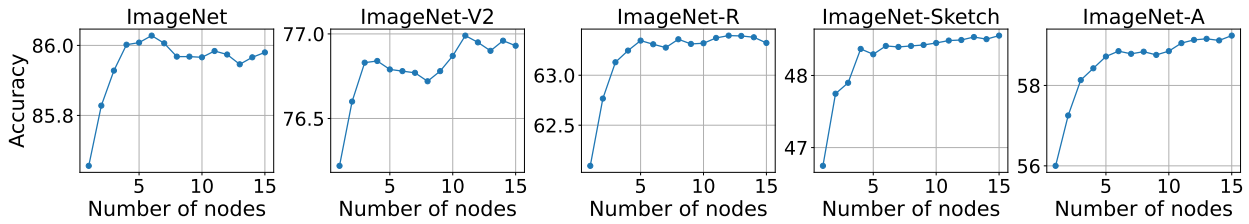

Figure 5.2: We test whether the performance of lo-fi continues to improve when adding more nodes. On the contrary, this experiment suggests diminishing or even negative returns after 4 nodes. This experiment is for fine-tuning DeiT-base as in Table 5.1. Recall that when using four nodes, lo-fi and the baseline observe the same number of images, but lo-fi does not require communication between nodes. When moving beyond 4 nodes as we do in this experiment, lo-fi observes more images then the baseline.

display the accuracy of the averaged lo-fi model at the end of each epoch, though usually we would only average the models once at the end. A question emerges when looking at this plot: why does the accuracy of the individual node first dip before coming back up? The answer is due to the interaction of learning rate and batch size, which we discuss further in the Appendix.

Table 5.1 evaluates the final models from Figure 5.1 on ImageNet as well as under distribution shift (on ImageNet-V2 [222], ImageNet-R [112], ImageNet Sketch [260], and ImageNet-A [113]). In addition, Table 5.1 repeats the experiment from Figure 5.1 with the DeiT-large model. We underline any result that is significantly better (using McNemar's test with significance 0.05). Overall we observe that lo-fi matches the accuracy of the baseline which uses communication, and outperforms the baseline under distribution shift.

Table 5.2 supplements Table 5.1 with additional details. In particular, we consider the accuracy of the model produced by an individual node during lo-fi, before the averaging. We also evaluate the output-space ensemble of the models produced by each node during lo-fi, which is more expensive during inference as a pass through each model is required. Finally, we display the accuracy of the models fine-tuned in the DeiT-III paper [255]. We improved our own baseline over that in the paper with the following hyperparemter changes: (i) Instead of removing the classification layer of the pre-trained model, we implement a version of LP-FT [148] to fine-tune—we preserved the ImageNet-21k classifier then use a class mapping from ImageNet-21k to ImageNet classes. (ii) We remove the grayscale, solarization, and Gaussian blur augmentations, since we found this improves accuracy. This aligns with previous research where fine-tuning requires less augmentation [273]. (iii) We fine-tuned for fewer epochs, which also required a switch to a cosine scheduler that updates every iteration instead of every epoch so the schedule could complete. We also considered

different values for the learning rate and stochastic depth, but found the default values to be best [255]. This is with the exception of DeiT-large for which we found stochastic depth 0.3 to be better for the baseline, which is what we used.

Lo-fi was run using identical hyperparameters except we decreased the stochastic depth drop rate by 0.05 for both DeiT-base and DeiT-large since each node is effectively fine-tuning on less data and may therefore require less regularization. The most substantive change from the DeiT-III code was to use LP-FT [148], which we accomplished by preserving the classification layer from the pre-trained model and using a mapping from ImageNet-21k to ImageNet[1]. While this change results in a minor improvement for the baseline, we found it was necessary for achieving matching performance with lo-fi. Overall, despite the extensive hyperparameter tuning we performed for the baseline, lo-fi was still able to match or exceed the accuracy.

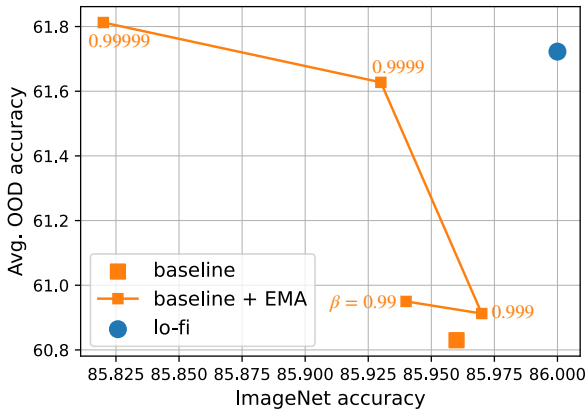

Figure 5.3: For the experiment in Table 5.1, lo-fi outperforms the baseline under distribution shift. We wanted to test whether this OOD performance (y-axis) could be improved by applying weight averaging techniques to the baseline. We observe that the answer is yes with EMA [249], although this can come at slight cost in-distribution accuracy (x-axis). In this plot we try 4 different values of EMA decay β. Applying EMA to lo-fi had minimal benefit, as did applying WiSE-FT [272] to the baseline. The ImageNet-21k→ImageNet transfer setting is not characteristic of those studied in the WiSE-FT paper.

Ablations. We ran three ablation studies to better understand the performance of lo-fi in this setting. First, we wanted to test whether adding more nodes was helpful. In the initial 4 node experiment with lo-fi, we matched the baseline in terms of total amount of data observed, allowing a fair compute-matched comparison. However, there are practical settings such as privacy-preserving ML in which the benefits of reduced communication may outweigh the importance of matched compute. In Figure 5.2 we observed that adding more nodes did not improve in-distribution accuracy. Interestingly, however, adding additional nodes marginally improved out-of-distribution performance, most notably on ImageNet-Sketch and ImageNet-A.

Next, we wanted to understand if four groups, one per node, was optimal in this setting. What happens if

[1]The only class in ImageNet but not ImageNet-21k is *teddy bear*—we initialize this row with *bear* instead.

Groups	2	4	8	16
ImageNet Accuracy	85.95	86.00	85.85	85.73

Table 5.3: For our four-node fine-tuning jobs, we usually partition the 32 GPUs into 4 communication groups, one per-node. This table shows the effect of partitioning the GPUs into groups of different sizes, finding slightly worse performance when the number of groups is large.

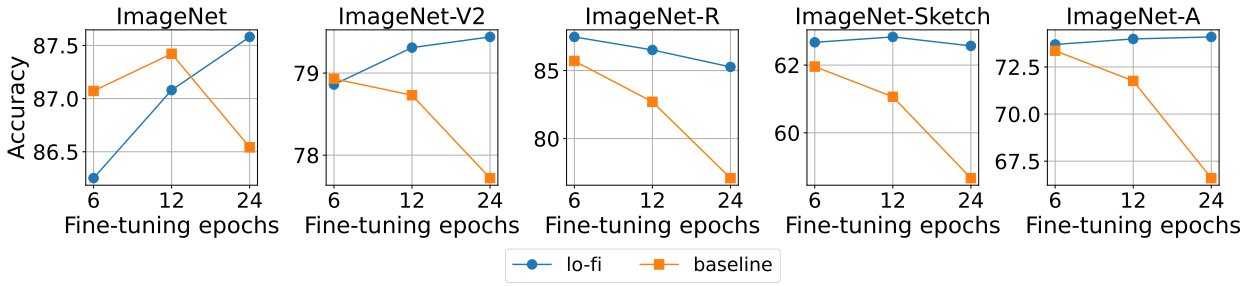

Figure 5.4: We fine-tune CLIP ViT-L [210, 71] on ImageNet. In contrast to the DeiT fine-tuning experiments, the models were not pre-trained with stochastic depth and we found better accuracy when fine-tuning without stochastic depth. Instead, we fine-tune for 6, 12, and 24 epochs. lo-fi shows good performance under distribution shift, but on ImageNet requires more epochs to exceed the baseline accuracy unlike in the DeiT experiments.

we instead use 8 groups—2 per node, or 2 groups—each group consisting of 2 nodes? In this experiments the amount of data observed remains constant; all that changes is the amount of communication. As presented in Table 5.3, accuracy drops slightly when using a larger number of groups[2]. This result demonstrates that the best configuration is one group per node.

Finally, we found it interesting the lo-fi outperformed the baseline under distribution shift. Accordingly, we wanted to test whether we could recover these out-of-distribution (OOD) improvements by applying other weight averaging techniques to the baseline. We observe in Figure 5.3 that the answer is yes, although at slight cost to in-distribution performance for the methods we tried. The best performing technique we tried was a debiased exponential moving average (EMA) [249, 136], for which we tried decay values 0.99, 0.999, 0.9999, and 0.99999. We also tried applying EMA and WiSE-FT [272] to lo-fi, but did not observe out-of-distribution improvements [3].

[2]When using 2, 8, and 16 groups we changed the stochastic depth drop rate by 0.05, -0.05, and -0.10, respectively, from the four group setting.

[3]The intuition from WiSE-FT [272] is that of combining a generalist and specialist. Our intuition for why WiSE-FT does not show substantial improvements in the ImageNet-21k→ImageNet transfer setting is because both models are ImageNet specialists.

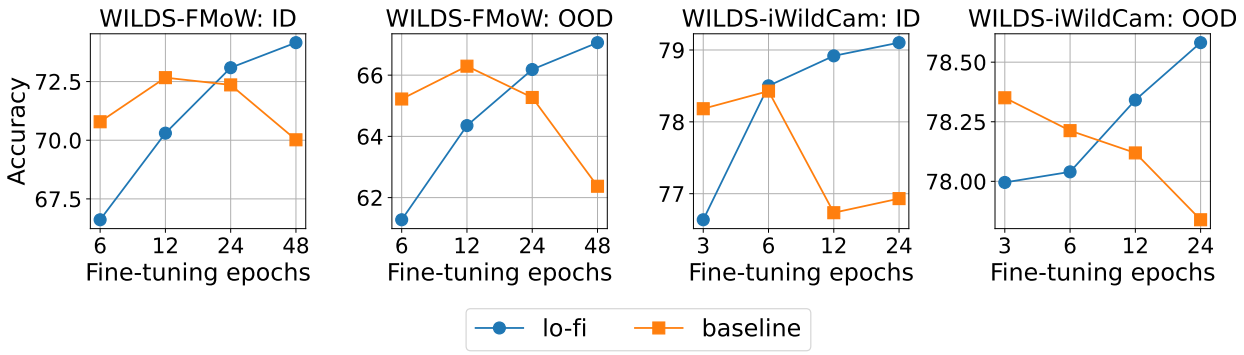

Figure 5.5: We repeat the CLIP ViT-L fine-tuning experiment from Figure 5.4 one two other image classification tasks: WILDS-FMoW [138, 45], a satelite recognition task with a geographic and temporal distribution shift and WILDS-iWildCam [138, 19], a camera trap dataset with a geographic distribution shift. Overall, we find similar results as in Figure 5.4.

5.4.2 Fine-tuning CLIP ViT-L on ImageNet and WILDS

In the previous section we observed that lo-fi matches the baseline for DeiT-III on ImageNet, but how does lo-fi perform for models pre-trained on larger datasets? In this section, we further test lo-fi for the CLIP ViT-L [210, 71] when fine-tuning on ImageNet (Figure 5.4) as well as two datasets from WILDS [138] (Figure 5.5).

Unlike the DeiT models, CLIP was not pre-trained with stochastic depth and we find better accuracy when we fine-tune without stochastic depth. This is unlike the DeiT-III models, which we found performed best when we used some stochastic depth. Indeed, this allowed us to use slightly less regularization for lo-fi then we did for the baseline by decreasing stochastic depth drop rate by 0.05. As this is no longer the case, we instead show experiments when fine-tuning for different numbers of epochs. Other than this omission of stochastic depth and varying the training epochs, the hyperparameter configuration is identical to that discussed in the previous section and follows the ImageNet-21k→ImageNet fine-tuning set-up from DeiT-III [255].

Results when fine-tuning CLIP ViT-L on ImageNet are presented in Figure 5.4. For this experiment, we initialize the classification head of the zero-shot model using the zero-shot classifier output by the CLIP text tower (as in Wortsman et al. [272]). We observe that more fine-tuning epochs are required for lo-fi to outperform the baseline on ImageNet. Under distribution shift, lo-fi roughly matches or exceeds the baseline for each of the fine-tuning epochs we tried. While this result indicates that lo-fi is a promising alternative to

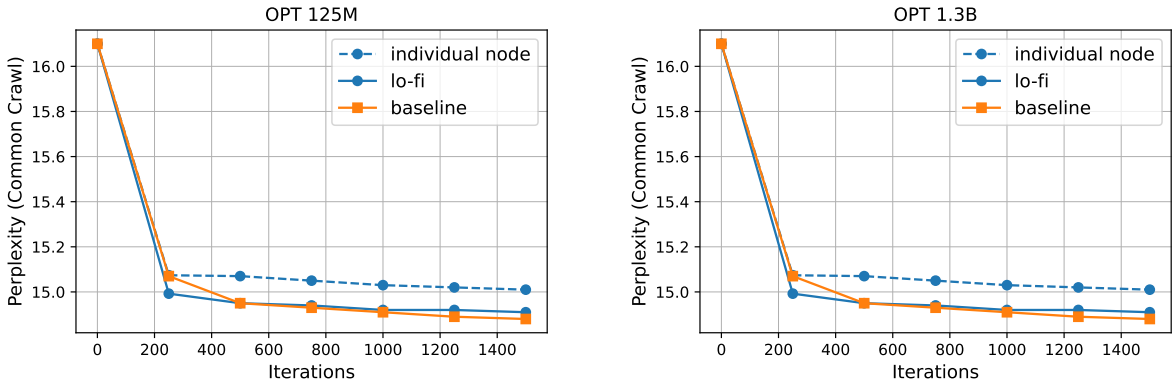

Figure 5.6: Fine-tuning a language model (left: OPT-125M, right: OPT-1.3B) on Common Crawl with lo-fi closely approaches the performance of the baseline of multi-node fine-tuning with communication. Here, we train four lo-fi workers independently, one per node. The baseline consists of standard data-parallel fine-tuning using four nodes, where there is communication between nodes at every iteration. The x-axis shows iterations, which does not take into account that lo-fi may be faster.

the baseline in this setting, a key limitation is that additional fine-tuning epochs were required to enable this improvement. The accuracy improvements beyond the best baseline model are consistent with the results reported in model soups [273].

We also test CLIP ViT-L on two further datasets, WILDS-FMoW [138, 45], a satellite image recognition dataset with a temporal distribution shift and WILDS-iWildCam [138, 19], a classification dataset with camera traps in the wild with a geographic distribution shift. Our motivation is to test lo-fi on natural images beyond the ImageNet universe. The results are presented in Figure 5.5, observing very similar results to the aforementioned experiment of fine-tuning CLIP ViT-L on ImageNet. However, there is an important difference in the experimental set-up. For these experiments, we first tried using the zero-shot initialization for the last layer of the model, as we did with ImageNet. However, this resulted in worse accuracy for lo-fi. Accordingly, these experiments are completed using the LP-FT method of fine-tuning [148]. First, we train a linear probe using one node. This linear probe is then used as the initialization when end-to-end fine-tuning individually on each node. We also apply this change to the baseline, but the benefit is much less substantial for the baseline than for lo-fi. Finally, for this experiment we used learning rate 7e-4 which we found resulted in higher accuracy for lo-fi and the baseline.

5.4.3 Language model fine-tuning

We also test lo-fi outside of image classification by fine-tuning OPT-125M and OPT-1.3B [301].

Experimental Setup We report i)individual node, which is the average performance of the models produced by each node when using lo-fi, ii) lo-fi, which averages models produced by each node, and iii) baseline, which uses communication between nodes. For the 125M parameter model, we set the learning rate to 6e-5, with 1024-length sequence blocks, and 500K tokens per batch. For the 1.3B parameter model, we set the learning rate to 1e-5, with 512-length sequence blocks, and 1M tokens per batch. We use fp16 mixed precision [183] for all experiments. We fine-tune the 125M parameter model with 4 nodes, and we fine-tune the 1.3B parameter model with 8 nodes. When using lo-fi there is no communication between nodes, so the experiments produce 4 and 8 models, respectively. Each node consists of 8 Volta 32GB GPUs connected with 400GBps interconnect.

Results We fine-tune on the Pile's Common Crawl subset [88] using the Huggingface Transformers library [268]. Results are presented in Figure 5.6. We observe that for both model scales, when comparing by step count, lo-fi roughly matches the performance of the baseline, providing large performance improvements over the individual node setting. These results suggest that lo-fi is an effective alternative to standard multi-node fine-tuning with communication.

5.4.4 How much is the speed-up, really?

We have shown that lo-fi produces high accuracy models without communication during fine-tuning. This leads to an important practical question: what is the wall-clock advantage of eliminating communication between nodes during fine-tuning? We examine the wall-clock training time advantage once nodes are allocated and also the time it takes for node allocation on a slurm cluster. Note that these experiments are for the DeiT-III [255] models in the image classification setting.

Wall-clock advantage. To examine the wall-clock advantage of lo-fi compared to the baseline we use A100 GPUs on AWS with fast interconnect of 400 GBps (EFA). This is representative of a fast and modern large scale neural network training set-up. In particular, we want to understand the effect of using modern

distributed training tools, and also varying batch size. We note that our results depend critically on the quality of the interconnect between nodes. In a setting with a slower interconnect such as standard ethernet, we would expect the training speed-ups to be more substantial. In a setting with a faster interconnect such as TPUs, the training speed-ups should be more minor.

A recent innovation in distributed training tooling is to overlap the backwards pass computation and gradient communication—the gradients for layer $\ell - 1$ can be computed at the same time as communicating the gradients for layer ℓ [161, 200] [4]. We experiment with turning on and off this overlapping communication/computation feature, finding substantial reductions in communication overhead when overlapping communication and computation. We also experiment with changing the batch size. In general, we observed that when using a smaller batch size, communication will account for a larger portion of training time. This is because the size of the gradient does not depend on the batch size, so absolute communication cost does not depend on batch size. However, using a smaller batch size will lower the total computation time and therefore communication cost will account for a larger fraction of the total training time.

Our experiments with varying batch size and turning on and off overlapping communication/computation are presented in Figure 5.7 (left). These experiments are for the vision transformer models DeiT-base, DeiT-large, DeiT-huge and DeiT-giant, ranging from roughly 10^8 to 10^9 parameters. On the x-axis we show the different model sizes, while the y-axis shows the additional wall-clock time required to go from a 1 node to 4 node job (i.e., 0% indicates that the 4 node job is the same speed as the 1 node job, while 100% indicates that the 4 node job is twice as slow). In this experiment, the number of iterations and batch size per device is fixed[5]. We found that without overlapping communication/compute, shifting to a multi-node settings results in a substantial increase in training time of 25-55% (purple lines). However, overlapping communication and compute has proven surprisingly effective, reducing the communication cost to $<$10%.

A potential issue with these experiments is that they currently reflects a "state-of-the-art" cluster setting, and actual workflows may be slower. We also believe that both GPU memory size and network bandwidth will improve in the future. Higher GPU memory capacity will allow users to train larger models, resulting in higher communication overhead, while higher network bandwidth will help to reduce the communication

[4] Overlapping communication/computation on by default in PyTorch $\geq$1.5 [200, 161].

[5] We note that scaling with fixed batch size may be unrealistic for certain problems as large batch sizes can cause accuracy to drop, which would be a reason to use lo-fi.

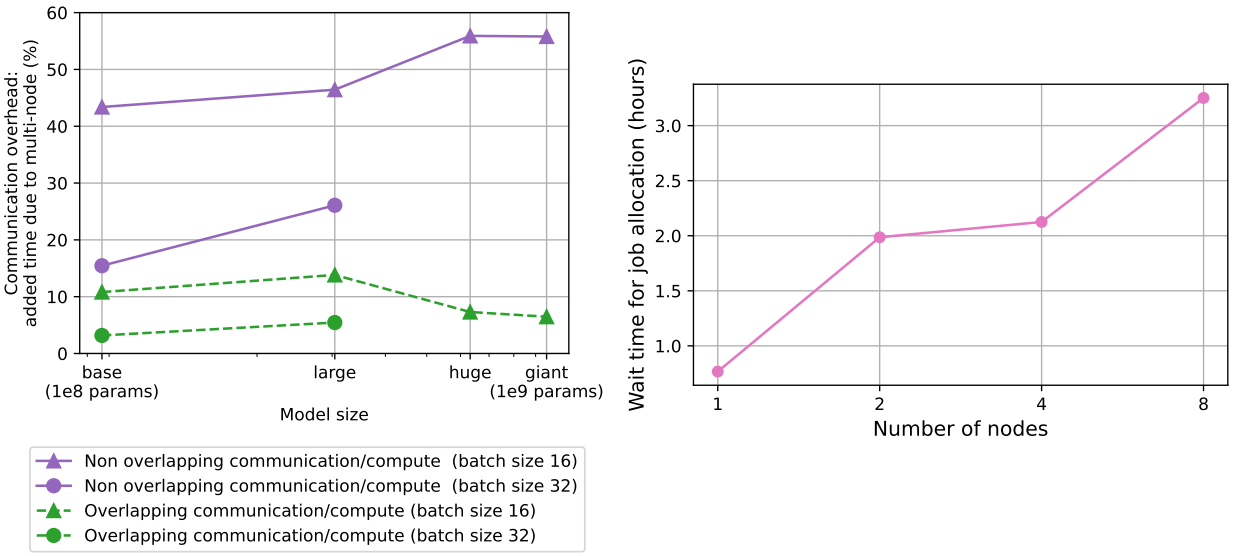

Figure 5.7: (Left) On an AWS cluster we show on the y-axis the wall-clock overhead observed when switching from 1 to 4 nodes using models from the DeiT-III repository [255] and constant per-GPU batch size. 100% indicates that the job becomes twice as slow while 0% indicates no difference switching from 1 to 4 nodes. With the method of overlapping the communication and computation in the backward pass [161], the slow-down is less substantial than we initially expected, especially for larger per-GPU batch sizes. The huge and giant models are deeper and there is more opportunity to overlap communication and computation. **(Right)** Jobs requiring only one node schedule faster than jobs requiring four nodes on the slurm cluster that we use for these experiments. This plot shows the median per-day wait time averaged over three months of job data on this cluster.

delay.

Finally, we note that lo-fi can help the straggler and jitter problem in distributed training, where different nodes might experience random slowdowns due to various reasons. In standard data-parallel, synchronization will take place multiple times per iteration, such that any random slowdown on any node will slow down the entire run. Since lo-fi needs only one communication at the end (which can even be asynchronous), the straggler/jitter problem is no longer an issue.

Scheduling advantage. For modern cluster workloads, both on private and public clusters, the wait time to schedule a job can increase the total training time, especially during periods of heavy cluster usage. Since single-node jobs require fewer simultaneous resources to run, they should schedule faster, reducing the total training time. To measure this, we analyzed the time required to schedule a 1-node job vs. a multi-node job on a large slurm-based cluster and present the results in Figure 5.7 (right). These wait times are averaged

over all jobs run on this cluster over a three month period. We found that scheduling a single node job was notably faster than multi-node jobs, taking ∼45 minutes for 1 node, ∼2 hours for 2-4 nodes, and ∼3 hours for 8 nodes.

We note that these results are specific to the cluster used in these experiments and may or may not be representative of other clusters depending on their scheduling algorithm and workload distribution, amongst other factors. We also note that the scheduling benefit will only apply when using implementation B in which each group is trained independently (as described in Section 5.3). Regardless, we thought that it may be useful to collect and present this empirical data, providing quantitative support for the observation from [160] that jobs requiring fewer nodes schedule faster.

5.4.5 Does jointly training to increase diversity across groups improve lo-fi performance?

Previous work from [101, 273] has shown that more diverse models trained with different hyperparameters produce larger benefits when ensembles or weight averaged and also [160] which showed that ensembling or weight averaging specialists trained on different domains incurs the largest benefit. We therefore asked whether encouraging diversity automatically through regularization during training might improve the performance of the final lo-fi model.

While this strategy did indeed produce models with a larger averaging benefit (avg. model - best individual model), it also decreased the accuracy of the individual models such that overall performance was the same or worse than simply training the lo-fi components independently. We also tried pulling together the predictions of the models, which is also known as co-distillation [7, 240]. This improved the accuracy of the individual models, but as model diversity decreased, the benefit from weight-averaging was reduced, also leading to overall lower accuracy. We explored a number of variations of these approaches which we discuss in more detail in the Appendix.

5.5 Related work

Averaging and linearly interpolating models. Averaging or interpolating the weights of neural networks is a common technique for improving accuracy.

Weight-averaging techniques for optimization date back to early work in convex optimization [227, 205].

In deep learning, an exponential moving average (EMA) of weights can be used to improve accuracy [249]. Another popular approach is Stochastic Weight Averaging (SWA) [123] which uses a uniform average of weights saved at each epoch while training with a constant or cyclic learning rate. Indeed, the SWA method was motivated in part by the analogy between weight-averaging and ensembling.

While SWA and EMA average weights along the training trajectory, there has also been substantial interest in averaging weights across independent trajectories. In particular, Nagarajan & Kolter [190] observe that the weights of two models that are fine-tuned independently on MNIST [153] from a shared initialization can be interpolated without increasing loss. For more difficult problems such as ImageNet, this naive linear interpolation encounters a high error barrier [82, 81]. However, Frankle et al. [82] observe that when the first part of the optimization trajectory is shared and the remainder of training is independent, models can once again be interpolated without reducing accuracy. They refer to this phenomena—interpolating weights without accuracy loss—as *linear mode connectivity*. Neyshabur et al. [192] observed a similar phenomenon when interpolating between model pairs that are *fine-tuned* from a shared initialization on a new task. This observation was extended to interpolation between a zero-shot model and fine-tuned model with the WiSE-FT approach [272], to many models fine-tuned with different hyperparameters with model soups [273], to models fine-tuned on different datasets with Ilharco et al. [121], and for creating better pre-trained models by Choshen et al. [43]. The overall observation that the objective landscape can appears roughly convex was also made by [158]. While all of the aforementioned weight-averaging employ simple linear interpolation, more advanced weight-averaging techniques have also been developed with promising results [176].

Recently, Li et al. [160] introduced branch-train-merge which is at the intersection of model combination and distributed training. They consider the case where the training data is partitioned into different textual domains, then train an individual expert model for each domain. As they are training from scratch, they first require an initial seed phase. They then combine all of these experts via weight averaging or ensembling to outperform the dense baseline of training one large model on all of the data. The main difference are that our work is for fine-tuning, and we do not assume the data is partitioned into different domains.

Other research in the area includes Garipov et al. [90] and Draxler et al. [73] who concurrently found that two neural network solutions trained independently can be connected by a simple curve along which loss remains low. These findings were generalized by Benton et al. [21] who learn high dimensional low-loss

connectors between individual solutions. Concurrent work with Benton et al. [21], Wortsman et al. [271] learned these high dimensional low-loss subspaces from scratch. Then, Entezari et al. [75] conjectured that all solutions could be made to be linearly connected by applying a permutation to the weights which does not change the function. Ainsworth et al. [3] recently made progress towards confirming this conjecture. However, unlike the model interpolations we observe here, and have previously been observed [273, 160], the interpolations in Ainsworth et al. [3] so far do not improve models in terms of accuracy. Regardless, they are interesting from a scientific perspective, and suggest the possibility of applying methods such as lo-fi for training from scratch in the future, although there is currently no evidence towards this.

Distributed training and fine-tuning. Distributed training [161, 288] and fine-tuning [28, 261] are increasingly important in deep learning as models become larger.

An overview of the many standard approaches is detailed by Weng & Brockman [266], including i) data-parallelism, where data is split among devices, ii) pipeline parallelism, where different layers are split among devices, and iii) tensor parallelism, where individual layers are split among devices. We note that these approaches are not mutually exclusive. Indeed, one can use pipeline parallelism to distribute a model across a node, then use data-parallelism across nodes. lo-fi is proposing an alternative to data-parallelism across nodes—instead of synchronizing the updates between nodes during fine-tuning, each node independently produces a model which is averaged at the end of fine-tuning. We emphasize that lo-fi can still be used if there is pipeline parallelism across the node.

There have previously been many alternatives proposed to synchronizing gradients each step. The idea of training several models in parallel and averaging their weights once at the end of training has been investigated at least since [180, 307]. The focus in those works is on convex models and training from scratch, rather than fine-tuning. While lo-fi is simply borrowing these techniques from convex optimization and applying them to fine-tuning for deep learning, we believe that these findings are interesting and useful from a practical standpoint.

Another alternative, HogWild [221] proposes asynchronous communication. The difference between HogWild and lo-fi is that lo-fi never communicates during fine-tuning, so it's as if the hogs each have their own individual farm. As another alternative, local-sgd [242, 196] communicates updates every k steps instead of every step. lo-fi is equivalent to local-sgd applied to fine-tuning where k is the number of fine-

tuning epochs.

There have also been compelling recent methods for more efficient and accessible pipeline or tensor parallelism to enable learning or inference with extremely large models. For instance, with Petals [28] certain layers of very large models are computed and communicated among coordinated users. Also, researchers have been using decentralized training for very large models [119, 288] which is made possible by, e.g., compressing communication [261, 131, 277, 78, 165]. Indeed, even inference with very many-billion parameter models can pose interesting challenges [62]. Our approach is orthogonal to the work in compressing communication, as we are instead removing communication, but may prove useful for large-scale decentralized training.

There is also the active research area of federated learning (e.g., Kairouz et al. [131], Pillutla et al. [203]), which has recently been explored in transfer settings [193]. In federated learning, the data on each client is different and updates are usually communicated every k steps. While lo-fi only considers the easier setting of IID data, it is possible that similar approaches based on weight averaging to reduce communication may prove beneficial for privacy-preserving machine learning.

5.6 Limitations and conclusion

Limitations. There are many limitations discussed throughout this text. For instance, we found that when fine-tuning CLIP ViT-L on ImageNet and WILDS, lo-fi needs to observe more data to exceed the baseline. This is similarly true during language model fine-tuning (Section 5.4.3). Therefore, we most recommend lo-fi when communication costs are prohibitive. A final limitation is that lo-fi can only achieve matching accuracy when no new parameters are introduced, which we accomplish with a "zero-shot" initialization, or via LP-FT (this does not come up during language model fine-tuning).

Conclusion. Overall we have observed that communication between nodes is not required during fine-tuning in certain settings. These findings may prove beneficial to a number of settings including large-scale decentralized fine-tuning and privacy-preserving ML and represent a promising step in the overall direction of developing models like open-source software [212] in which many institutions can collaboratively fine-tune a large model if none has the resource to do so individually. As more workloads shift to fine-tuning of pre-trained models and models grow increasingly larger, we hope that our results will help to reduce barriers

to large-scale models.

Chapter 6

Stable and low precision training of large-scale vision-language models

6.1 Overview

We introduce new methods for 1) accelerating and 2) stabilizing training for large language-vision models. 1) For acceleration, we introduce SwitchBack, a linear layer for int8 quantized training which provides a speed-up of 13-25% while matching the performance of bfloat16 training within 0.1 percentage points for the 1B parameter CLIP ViT-Huge—the largest int8 training to date. Our main focus is int8 as GPU support for float8 is rare, though we also analyze float8 training through simulation. While SwitchBack proves effective for float8, we show that standard techniques are also successful if the network is trained and initialized so that large feature magnitudes are discouraged, which we accomplish via layer-scale initialized with zeros. 2) For stability, we analyze loss spikes and find they consistently occur 1-8 iterations after the squared gradients become under-estimated by their AdamW second moment estimator. As a result, we recommend an AdamW-Adafactor hybrid which avoids loss spikes when training a CLIP ViT-Huge model and outperforms gradient clipping at the scales we test.

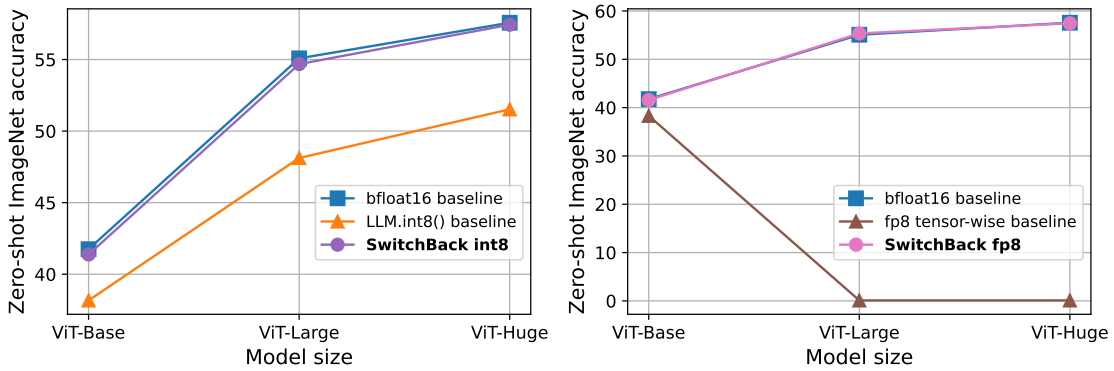

Figure 6.1: We introduce SwitchBack, a linear layer for low-precision training. **(Left)** SwitchBack for int8 training matches the zero-shot ImageNet [59] accuracy of standard bfloat16 training within 0.1 percentage point for CLIP ViT-Huge [210, 71] and outperforms LLM.int8() [62]. **(Right)** For float8 (fp8) training [184], a baseline which uses tensor-wise quantization diverges for large models while SwitchBack matches the baseline. In these large-model, small-data experiments, our focus is on comparing methods and not final model accuracy, so we use short runs which makes it feasible to run many experiments.

6.2 Introduction

Large models trained on large datasets have recently led to multiple breakthroughs in machine learning such as GPT-3 [33] and PaLM [44]. While many components are necessary for successful large-scale training, two critical elements are training speed and stability. To enable further progress, we must ensure that 1) training is fast—the model should be able to see a lot of data even if it is large, and 2) training is stable—large models should not suffer from loss spikes which degrade performance. We study these two directions in the context of contrastive language-image pre-training (CLIP) [210]. We examine CLIP-style models because of their importance in computer vision: CLIP-style models reach state-of-the-art performance on a wide range of image classification tasks [210, 273, 202, 38] and underlie image generation methods such as DALLE·2 [218] and Stable Diffusion [226]. Our contributions towards fast training and stable training are as follows.

Towards fast training, we introduce **SwitchBack**, a linear layer for quantized training with int8 precision which matches the performance of the bfloat16 [263] baseline within 0.1 percentage points for CLIP ViT-Huge—a larger model than considered in the original CLIP paper. Linear layers account for the majority of the compute in standard transformer models, usually more than 90%, comprising the key, query, value, and out projection of the attention blocks as well as the multilayer perceptron. We perform all linear

layers in low-precision (int8) while retaining other layers, such as layer norms, in higher precision. With this setup, we observe end-to-end speedups between 13 and 25% for CLIP ViT-Huge training: 25% compared to a standard linear layer implemented using the PyTorch [200] autograd python module and 13% compared to the standard PyTorch layer which include background CUDA/C++ optimizations which are difficult to replicate for custom layers.

SwitchBack starts from the observation that quantization noise grows with the inner dimension in a matrix multiplication. For CLIP training, the weight gradient computation involves a large inner dimension because CLIP training requires a large batch size [202]. Hence SwitchBack uses 16 bit precision matrix multiplication for the weight gradient computation while using int8 multiplications for the forward pass and layer input gradient computations. This approach leads to large accuracy improvements compared to LLM.int8() [62] (Figure 6.1). We will provide open-source Triton [252] kernels for Switchback to enable future work on efficient quantization schemes.

Besides int8 training, we also study large-scale 8-bit float (fp8) [184] training. We do not have access to hardware that supports fp8 data types, which is currently more rare than int8, so we use an accurate simulation of fp8 computation. SwitchBack also outperforms straightforward 8-bit float (fp8) baselines because tensor-wise quantized baselines diverge at $>$420M scale (Figure 6.1). However, we demonstrate that these methods can achieve high accuracy if the network is trained while keeping feature magnitudes small, which we accomplish via layer-scale [254] initialized with zeros.

Towards stable training, we find that loss spikes occur in CLIP training when the AdamW [169] second moment estimator becomes out-of-date in the patch embedding [71] layer. In particular, the learning signal changes so that the moving averages of squared gradients underestimates their true magnitude. Indeed, in the absence of stability interventions, we show that loss spikes can be predicted by examining this ratio of the squared gradients to their moving average. We therefore recommend an AdamW-AdaFactor [234] hybrid, which we refer to as **StableAdamW** as it removes instabilities at the scales we consider and outperforms gradient clipping. Concretely, StableAdamW is AdamW with the update clipping technique introduced in AdaFactor. Update clipping tracks the average ratio of the gradient square to the second moment estimator and lowers the learning rate when the ratio is large.

The remainder of this paper is organized as follows: Section 6.3 focuses on low-precision training while

Section 6.4 stabilizes training by reducing loss spikes.

Algorithm 6.1: PyTorch code for **SwitchBack**

```
class SwitchBackMatmul(autograd.Function):
   @staticmethod
   def forward(ctx, X, W):
      ctx.save_for_backward = X, W

      X_int8, state_X = row-wise_quantize(X)
      W_int8, state_W = tensor-wise_quantize(W)

      return matmul_int8_and_dequanitze(
         X_int8, W_int8.t(), state_X, state_W
      )

   @staticmethod
   def backward(ctx, G):
      X, W = ctx.save_for_backward

      G_rowwise = rowwise_quantize(G)
      W_int8, state_W = tensor-
           wise_quantize_transpose(W)

      X_gradient = matmul_int8_and_dequanitze(
         G_int8, W_int8.t(), state_X, state_W
      )
      W_gradient = matmul_fp16(G.t(), X)

      return X_gradient, W_gradient
```

Algorithm 6.2: StableAdamW $(\{\alpha_t\}, \beta_1, \beta_2, \epsilon)$

$v_0, u_0 = \mathbf{0}$
for $t = 1$ **to** T **do**
  $g_t = \nabla f(\theta_t)$
  // apply correction term to debias EMA.
  $\hat{\beta}_1 = \beta_1 \cdot \frac{1-\beta_1^{t-1}}{1-\beta_1^t}$
  $\hat{\beta}_2 = \beta_2 \cdot \frac{1-\beta_2^{t-1}}{1-\beta_2^t}$
  // update moving averages
  $v_t = \hat{\beta}_1 v_{t-1} + (1-\hat{\beta}_1) g_t$
  $u_t = \hat{\beta}_2 u_{t-1} + (1-\hat{\beta}_2) g_t^2$
  // for implementation convenience
  // operations below occur independently for each tensor
  $\mathsf{RMS}_t = \sqrt{\mathbb{E}\left[g_t^2/u_t\right]}$
  // update parameters
  $\eta_t = \alpha_t / \mathsf{max}\left(1, \mathsf{RMS}_t\right)$
  $\theta_t = \theta_{t-1} - \eta_t \lambda \theta_{t-1} - \eta_t v_t / \left(\sqrt{u_t} + \epsilon\right)$

6.3 8-bit training

This section develops and compares methods for eight-bit training of languge-vision transformer models. First, Section 6.3.1 discusses preliminaries and related work. Next, Section 6.3.2 introduces and tests SwitchBack, a linear layer for int8 and float8 training. Finally, Section 6.3.3 develops alternatives to SwitchBack which can be used for float8.

6.3.1 Preliminaries and related work

Neural networks today typically use 16-bit operations for training [183] in either the float16 or bfloat16 format [263]. Floating point formats use a subset of bits to represent the exponent while the remainder specifies the fraction (often referred to as the mantissa). The float16 format uses 5 bits for the exponent while bfloat16 uses 8 and therefore covers a larger range—float16 has a range of $(5.96 \cdot 10^{-8}, 65504)$ while bfloat16 has a range of $(10^{-38}, 3 \cdot 10^{38})$. Most floating point formats also have denormalized numbers which allow for a "soft underflow" which gets exponentially closer to 0.0f for each additional bit in the mantissa. To

prevent underflows float16 mixed precision training [183] has been developed which works as follows. The loss of a mini-batch is multiplied by a loss scalar to scale the loss and following backpropagation gradients into the representable range of fp16. This loss scaling is undone by rescaling the weight gradients before the optimizer updates fp32 main weights with the fp16 gradients. In PyTorch [200], the loss scalar is initialized to 65536. Everytime an Inf/NaN is encountered, the update is skipped and the loss scalar is halved. If no Inf/NaN are encountered for 2k iterations, the scalar is doubled.

When the loss scalar becomes too low in float16 training the loss slowly diverges. This was observed by Cherti et al. [41] when training ViT-Huge CLIP models and remedied by switching to bfloat16. Another instance of float16 creating issues at scale was the training of OPT [301]. Indeed, many obstacles faced during the OPT project could have been alleviated by using bfloat16 [300]. Similarly, all float16 training runs for BLOOM ended in divergence, only after using bfloat16 was the training stable. However, fast bfloat16 support is only available on TPUs, or GPUs developed with or after the NVIDIA Ampere series (2021 or later).

While 16 bit training is the standard today, hardware support for 8 bit operations are becoming more common. Hopper GPUs support float8 (fp8) [184] and Ampere GPUs support int8. However, it is currently (2023) very difficult to attain Hopper GPUs. Moreover, while int8 and int4 are used for inference [62, 274, 61], and there is earlier work exploring 8 bit training for convnets [262, 306, 42], these formats are not commonly used for training transformer models at scale. The CLIP ViT-Huge models we train have 1B parameters including the image and text towers which is 40x larger than a standard ResNet-50 (23M) [107], and quantization is more challenging for large tensors [62].

6.3.2 SwitchBack

Method

Overview. A linear layer consists of three matrix multiplications—one in the forward pass to compute outputs and two in the backwards pass to compute gradients for the input and weights. Our SwitchBack layer uses 8 bit precision for the first two matrix multiplies but switches back to higher precision for the weight gradient.

We compute the weight gradient in higher precision because this matrix multiplication involves dot

products between vectors which have a length of batch size times sequence length. As CLIP training requires large batch sizes [210, 202], this inner dimension of batch size times sequence length is much larger than for the other matrix multiplies. Variance due to quantization increases with the inner dimension of the matrix multiply. This modification is what differentiates SwitchBack from LLM.int8(), allowing SwitchBack to match the bfloat16 baseline (Figure 6.1).

Notation. A standard linear layer is comprised of inputs $X \in \mathbb{R}^{b \times n}$, weights $W \in \mathbb{R}^{m \times n}$, and outputs $Y \in \mathbb{R}^{b \times m}$. In the forward pass, outputs are computed as $Y = XW^\top$. In the backwards pass the layer receives gradients of the loss with respect to Y, which we denote $\dot{Y}$. Then, gradients to inputs $\dot{X}$ are computed via $\dot{X} = \dot{Y}W$ while gradients to the weights $\dot{W}$ are computed via $\dot{W} = \dot{Y}^\top X$. For linear layers in a transformer [257], b is batch size times sequence length, while n and m are small multiples of the embedding dimension.

Quantization. For the matrix multiplies in 8 bit precision we use quantization. There are a multiple quantization techniques to choose from and we will release code for all these alternatives. However, we find the best trade-off of simplicity and performance is from using i) row-wise quantization [135] for the inputs and gradients and ii) tensor-wise quantization for the weights. Additional information on quantization methods is provided by Dettmers et al. [62] but we summarize below. Using int8 as an example, which can represent integers from -127 to 127, we now define row-wise and tensor wise quantization. For a matrix X with rows $x_1, ..., x_b$, row-wise quantization Q_{row} and tensor-wise quantization Q_{tensor} are given respectively by

$$Q_{\text{row}}\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \text{round}\left(\begin{bmatrix} \frac{127}{\text{absmax}(x_1)} \cdot x_1 \\ \vdots \\ \frac{127}{\text{absmax}(x_b)} \cdot x_b \end{bmatrix}\right), \quad Q_{\text{tensor}}(X) = \text{round}\left(\frac{127}{\text{absmax}(X)} \cdot X\right) \tag{6.1}$$

where absmax is the maximum of the absolute value.

Importantly, when applying Q_{row} we also save the row-wise absolute maximums so that we can use them later for dequantization. We refer to this as the quantization state, or *state*, for short, so $\text{state}_{\text{row}}(X) = [\text{absmax}(x_1), ..., \text{absmax}(x_b)]^\top \in \mathbb{R}^{b \times 1}$. Equivalently, for tensor-wise quantization we only need to store the tensor-wise absolute maximum so $\text{state}_{\text{tensor}}(X) = \text{absmax}(X) \in \mathbb{R}$.

Since only the matrix multiply occurs in int8 precision we need to dequantize the outputs back to the original floating point precision. The forward pass with quantization and dequantization becomes

$$\frac{\mathsf{state}_{\text{tensor}}(W)}{127^2} \cdot \mathsf{state}_{\text{row}}(X) * \underbrace{Q_{\text{row}}(X)\, Q_{\text{tensor}}(W)^{\top}}_{\text{int8 matmul}} \tag{6.2}$$

where $*$ denotes elementwise-multiplication, which in this case is broadcasted so that row i of the matrix $Q_{\text{row}}(X)\, Q_{\text{tensor}}(W)^{\top}$ is multiplied by element i of $\mathsf{state}_{\text{row}}(X)$.

As mentioned previously, we use row-wise quantization for the inputs and gradients and tensor-wise quantization for the weights. We find that using row-wise quantization for both matrices increases complexity at a negligible or no performance increase. As such, we use this simpler approach.

The last detail in our algorithm is hardware specific. NVIDIA GPUs, which we use in this work, do not implement the int8/float8 operation AB for matrices A and B and only AB^T is implemented. As such, it is necessary to transpose the weight matrix in the backward pass. To reduce the overhead of transposition and quantization we fuse both operations, meaning we load the required data once from slow DRAM into fast SRAM/shared memory and then perform both operation in this cached memory – this is critical for achieving speedups. We call this operation `tensor-wise_quantize_transpose`, which is a fused tensor-wise quantize and transpose operation. Putting the pieces together, the result is Algorithm 6.1.

Variants. While Algorithm 6.1 is the most straightforward version of SwitchBack, we also present two alternative versions—SwitchBackM and SwitchBackQ—and will release triton [252] implementations for all three. SwitchBackM is a memory efficient version of SwitchBack which only saves 8 bit tensors for the backwards pass—we recommend its use when memory is limited. The small downside of SwitchBackM is that it requires an additional dequantize operation during the backwards pass which increases the runtime overhead. For CLIP ViT-Huge we observed only a negligible accuracy differences between SwitchBack and SwitchBackM. In addition, we present SwitchBackQ which uses row-wise and column-wise quantization for the weights instead of tensor-wise. While we did not observe this to improve accuracy at the scales we consider, it's possible that it will perform better than SwitchBack at larger scale.

float8. While the explanation so far has used int8 as an example, the code for SwitchBack and float8 (fp8) is nearly identical. The only modification is that operations such as $\mathsf{round}(127x/\mathsf{absmax}(x))$ are

replaced by $\mathsf{float8cast}(x/\mathsf{absmax}(x))$ where we simulate float8cast by rounding to the exact values of the float8 data type. This simulation improves on the simulation of [184] which only clips the input tensors into the representable range of the float8 data type, but not the exact values of the float8 data type. This simulation theoretically matches float8 training, but we are unable to perform real float8 training because we lack the hardware that supports float8 arithmetic. As such, we perform arithmetic in 16-bit with exact float8 values. For our int8 experiments we conduct the multiplications in int8 using A100 GPUs—we perform real int8 training without any simulation.

Experimental setup

To evaluate SwitchBack we train CLIP [210] visual transformer [71] models on LAION-2B [230]. Typically CLIP training, especially at ViT-Huge scale, is prohibitively expensive. Our goal is not high final accuracy but rather to contrast different methods for low-precision training. To enable running multiple experiments, we therefore only train for a small number of samples seen—380 million images—and use patch-dropout 0.5 [163]. We note that the experiment is still very expensive, corresponding to roughly 300 epochs of ImageNet training in terms of samples seen, or approximately 2.9e20 FLOPs per training run. After training on LAION-2B we evaluate the models zero-shot on ImageNet [59] using the 80 prompt templates from CLIP [210].

We use batch size 16384 (per-gpu batch size of 256) and train for a total of 20k iterations. The first 5k iterations are linear warmup while the remaining 15k are cosine decay. Training and evaluation are conducted with the OpenCLIP library [120] with learning rate 2e-3, weight decay 0.2, and batch-size 16384 using the optimizer described in Section 6.4.5.

Results

We test two main questions: (1) can we replicate 16-bit performance with SwitchBack and (2) can we get speedups. To test (1) we train CLIP models with SwitchBack across multiple scales with both int8 and float8 precision (Figure 6.1). To test (2) we profile operations in an individual linear layer and also measure end-to-end training speed.

Accuracy. We find that SwitchBack can match standard 16-bit training performance and outperform

baselines for both a) int8 precision and b) float8 precision.

For our int8 experiments (Figure 6.1, right), we contrast the performance of i) the standard baseline which uses mixed-precision bfloat16, ii) the matrix multiplication kernels from LLM.int8() [62], which is equivalent to SwitchBackQ if the weight gradient multiplication was also performed in int8 using row- and column-wise quantization, and iii) SwitchBack. SwitchBack has a negligible accuracy drop of 0.1 percentage points compared to the bfloat16 baseline for CLIP ViT-Huge. In contrast, there is a drop of 5.9 percentage points when training with LLM.int8().

For our simulated float8 training experiments (Figure 6.1, right), we contrast the performance of i) the standard baseline which uses mixed-precision bfloat16, ii) a baseline which uses tensor-wise quantization for all matrices, that is the weights, inputs, and gradients, and iii) SwitchBack. SwitchBack has a negligible accuracy drop of 0.1 percentage points from the bfloat16 baseline for CLIP ViT-Huge. In contrast, training diverges for the baseline that uses tensor-wise quantization for all matrices.

Speed. By writing custom triton kernels [252] we achieve end-to-end speedups from 13-25% for CLIP ViT-Huge training. 25% compared to a standard linear layer implemented using the PyTorch [200] autograd python module and 13% compared to the standard PyTorch layer which include optimizations that are difficult to replicate for custom layers. Moreover, we find that the overhead due to quantization operations decreases with scale and is $\sim$10% for CLIP ViT-Huge.

6.3.3 Float8 training by reducing feature magnitude

We find that SwitchBack is necessary for high accuracy int8 training. However, this section develops other interventions which enable float8 training without SwitchBack. We show that high accuracy can be achieved via float8 training with tensor-wise quantization for the inputs, weights, and gradients, so long as the network is initialized and trained in a way which discourages large feature magnitudes. We accomplish via layer-scale [254] initialized to zero.

We use the bitsandbytes library [63] to simulate float8 training using the fp8 types from Micikevicius et al. [184]. We use tensor-wise quantization for the inputs, weights, and gradients, so that all operations occur in simulated float8. In our simulation, we represent each value only with the exact values representable by float8, but we perform computations in float16 precision. We believe that tensor-wise quantization ap-

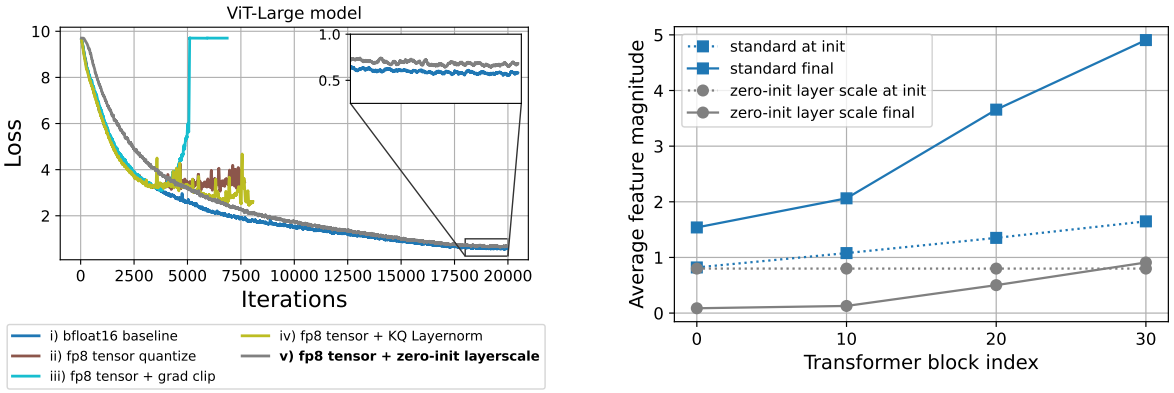

Figure 6.2: (Left) Training CLIP ViT-Large models with simulated fp8 precision using tensor-wise quantization for the inputs, weights, and gradients. All methods we try diverge except for using *zero-init layer-scale* [254], which multiplies the output of each self-attention or mlp block with a learnable vector initialized to zero. **(Right)** Examining feature magnitudes (i.e., the average absolute value of the output for transformer block k) for CLIP ViT-Huge at the beginning (init) and end of training. This suggest why zero-init layer scale enables float8 training—zero-init layer scale prevents high feature magnitudes which may cause issues for low precision training [62]. Without the intervention, the average feature magnitude becomes large for later blocks.

proximates the removal of quantize operations entirely. This is because the maximum of these tensors tends to evolve smoothly. Consequently, using a moving average for a maximum which is divided directly in the matmul is similar to tensor-wise quantization.

Layer-scale, introduced by Touvron et al. [254], scales each self-attention and MLP block output hidden state by a learnable vector of shape embed_dim. A pre-norm transformer block with layer-scale tensors γ_1 and γ_2 is defined as

$$x'_k = x_k + \gamma_1 * \mathsf{self\_attention}(\mathsf{norm}_1(x_k)), \quad x_{k+1} = x'_k + \gamma_2 * \mathsf{mlp}(\mathsf{norm}_2(x'_k)), \tag{6.3}$$

where $*$ is broadcasted elementwise multiplication.

Typically, layers are initialized so that they approximately preserve the variance of their inputs, and inputs have approximately unit variance [98, 106]. However, when combined with residual connections this can lead to higher norms in deeper networks.

Consequently, researchers have proposed initialization and scaling schemes which remedy this issue [13, 297, 32, 67]. Layer-scale with initialization 0 is an example of one such scheme—at initialization the transformer is an identity function. While γ_1, γ_2 are typically initialized as vectors of 10^{-4} or 10^{-6}, we use

0 for simplicity.

Figure 6.2 (right) demonstrates that the layer-scale intervention is successful at controlling the average magnitude output. Without the intervention, the average feature magnitude $\mathbb{E}[\mathsf{abs}(x_k)]$ becomes high for later blocks. Previous work [62] has shown that large feature magnitudes result in issues for low precision training.

Results for simulated fp8 training are shown in Figure 6.2 (left) for ViT-Large. We find that all fp8 runs diverge except for when we use layer-scale initialized to zero. Concretely, Figure 6.2 compares i) the baseline which uses bfloat16 training, ii) using fp8 with tensor-wise quantization and no further modifications, which slowly diverges, iii) adding gradient clipping to ii), which also diverges, iv) adding KQ layernorm [58] to ii), which also diverges, and v) using *zero-init layerscale*, which trains without diverging. While there is a difference still between fp8 and bfloat16 training, this is primarily because of layerscale. Moreover, we believe that with hyperparameter tuning layerscale would match standard training in terms of accuracy.

6.4 Stability

We now switch focus from accelerating learning by reducing precision to addressing instabilities which can arise during training. Section 6.4.1 reviews preliminaries and related work while Section 6.4.2 details the experimental setup. Next, Section 6.4.3 examines trends for training instability, finding loss spikes to increase with model scale but decrease with lower AdamW β_2. Then, Section 6.4.4 finds that loss spikes arise in our setting due to an out-of-date AdamW second moment estimator leading Section 6.4.5 to adopt and tests a fix developed in the context of AdaFactor [234].

6.4.1 Preliminaries and related work

Loss spikes can emerge when scaling up models [39, 95, 58, 291, 295, 234, 301]. These instabilities may slow learning, or even destabilize training completely. Various solutions have been proposed, including freezing the embedding layer [39], adding additional layer normalization [58, 95], or reparametrizing the weights [291].

In our work we investigate instabilities which arise during CLIP training. Unlike the instabilities observed in [58, 291] which lead to a slow divergence, we study fast loss spikes. Our results indicate that these

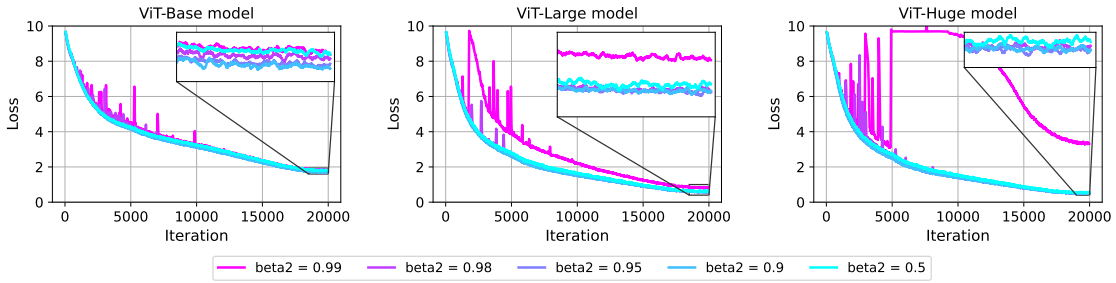

Figure 6.3: Loss spikes increase with **model size** for fixed learning rate and batch size. Reducing AdamW β_2 from its default in PyTorch of 0.999 mitigates loss spikes. Reducing β_2 too much slows training.

spikes arise when the second moment estimator is out of date for early layers.

While our analysis and methods build directly on Shazeer and Stern [234] (AdaFactor), there are important differences. In contrast with Shazeer and Stern [234], who only observe instabilities without warmup, we observe instabilities despite a long warmup period. Moreover, in contrast with Shazeer and Stern [234] we find that an out-of-date second moment estimator is primarily an issue for the (patch) embedding layer, and measure how well loss spikes are predicted by this event. Finally, we note that researchers have moved away from AdaFactor in its original formulation for large-scale training [211, 44, 292], finding AdaFactor to under-perform AdamW [211]. We believe this is due to the factored second moment or absence of first moment. This is why our focus is AdamW [169] which is the de facto standard optimizer for transformers.

6.4.2 Experimental setup

As in Section 6.3, we train ViT CLIP models on LAION [230] using OpenCLIP [120] and evaluate them zero-shot on ImageNet. Since we are not interested in final performance and instead interested in studying instability—even for very large models—we use a short run which allows us to conduct multiple experiments. Concretly, we use patch-dropout 0.5 [163] and 20k iterations. The first 5k iterations are linear warmup while the remainder are cosine decay [168]. We follow the CLIP paper [210] in that i) we do not use gradient clipping unless otherwise mentioned, though we do clip the logit_scale parameter, and ii) we add a layer-norm after the patch embedding and before the main transformer. Unless otherwise mentioned, experiments use batch size 16384 (per-gpu batch size of 256), learning rate 2e-3 and weight decay 0.2. We initially tried adding a layer-norm before the patch embedding as in [149], but removed this as we found it to hurt performance at CLIP ViT-Huge scale.

6.4.3 Loss spikes increase with model size, batch size, and learning rate

We begin our studying of loss spikes by observing how their presence varies when changing model size, batch size, and learning rate. The following sections build on these observations—in particular the finding that lowering the AdamW β_2 hyperparameter removes spikes entirely.

We find that loss spikes increase when increasing model size, batch size, or learning rate. The first result is shown in Figure 6.3 while the second two analagous results are in the Appendix. Importantly, these figures show that loss spikes can be avoided by reducing the β_2 hyperparameter for in AdamW. On the other hand, if β_2 is reduced too much then learning is slowed which results in worse performance [223].

6.4.4 On β_2 and an out-of-date second moment estimator

Based on the observation in the previous section that lowering β_2 reduces spikes, this section traces the cause of loss spikes to an out-of-date second moment estimator in the patch embedding layer.

Overview. Adaptive optimizers such as AdaGrad [74], Adam [136], or AdaFactor [234] scale the update differently for each individual parameter. This is often conceptualized a per-parameter learning rate. For instance, in Adam/AdamW, per-parameter updates are scaled by the inverse root of the exponential moving average of squared gradients (see the code for AdamW in Algorithm 6.2, ignoring for now the modifications in pink which we discuss in Section 6.4.5).

This adaptivity can be a very useful tool for accelerating training, but can also cause issues when the learning signal changes. Concretely, exponential moving averages can become out of date causing updates to be scaled by a value that is too large. This issue is discussed in Section 5 of Shazeer and Stern [234], and we summarize below.

As in Algorithm 6.2, let $u_t = \{u_{t,j}\}_{j=1}^n$ denote the exponential moving average (EMA) of squared gradients $g_t^2 = \{g_{t,j}^2\}_{j=1}^n$ for neural network parameters $\theta \in \mathbb{R}^n$. Ignoring the bias correction term[1], at each iteration t, u_t is updated as $\beta_2 u_{t-1} + (1-\beta_2)g_t^2$ where β_2 is referred to as the *decay* for the EMA. Then, the update is scaled by $1/\left(\sqrt{u_t}+\epsilon\right)$, where ϵ is a small value added numerical stability. Often the ratio $v_t/\left(\sqrt{u_t}+\epsilon\right)$ is thought of as signal-to-noise ratio of the gradient over time.

[1] In practice, the EMA is debiased with a correction term. Algorithm 6.2 follows AdaFactor section 7.1 in applying the correction term to β_1, β_2. Adam is often written with the correction term applied to v_t, u_t but they are equivalent [234].

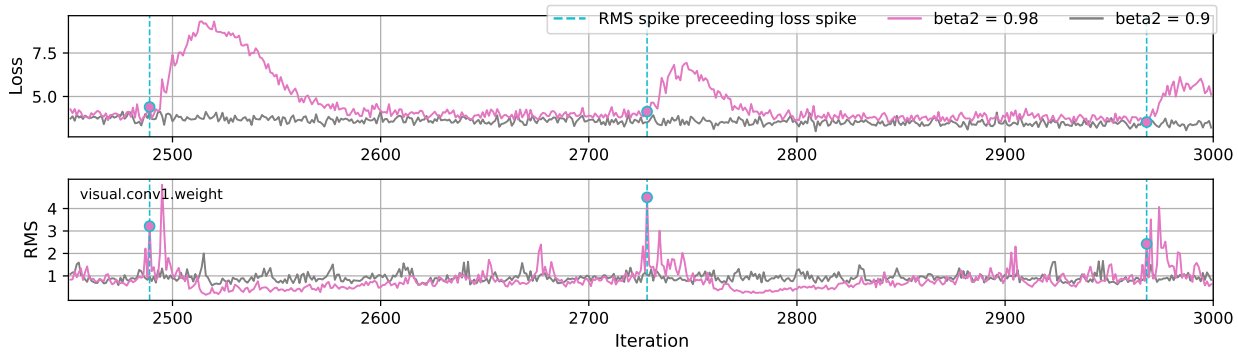

Figure 6.4: The learning signal can change so that the AdamW second moment estimator u_t is out-of-date and underestimates the squared gradients g_t^2. This can be detected if the aggregate quantity $\mathsf{RMS}_t = \sqrt{\mathbb{E}\left[g_t^2/u_t\right]}$ is far from 1. This figure observes a predictive relationship between the event of an RMS spike and a loss spike— we observe a spike in RMS_t 1-8 iterations before a loss spike. For lower β_2, RMS_t does not deviate far from 1. This result looks at RMS_t for the patch embedding layer only. This predictive relationship is further examined in the Appendix.

However, this method can break down when the learning signal changes and u_t ceases to be a good estimator for the running average of g_t^2. Consider the case where the gradient magnitudes have been historically very small for some parameters so $1/\left(\sqrt{u_t}+\epsilon\right)$ is large for those parameters. If, then, at iteration t those parameters suddenly receive a larger gradient signal the update can be catastrophically big. We refer to the scenario as the **stuck-in-the-past** scenario.

Overall, if β_2 is too small then convergence may be slowed [223]. If β_2 is too large then u_t can become out-of-date and no longer a good estimator for g_t^2, resulting in per-parameter scaling that is too large.

Measurement. We now discuss measurement of the aforementioned **stuck-in-the-past** scenario and search for a predictive relationship between this event and a loss spike. We follow Shazeer and Stern [234] and measure the following root-mean-square quantity, $\mathsf{RMS}_t = \sqrt{\mathbb{E}\left[g_t^2/u_t\right]}$. If u_t is a good estimator for g_t^2 then the aggregate quantity RMS_t will be around 1. The **stuck-in-the-past** scenario described above corresponds to an $\mathsf{RMS}_t \gg 1$.

As illustrated in Figure 6.3 we observe instability for high β_2 in our experiments even though we have 5k iterations of warm-up. While Shazeer and Stern [234] first recognize the out-of-date second moment estimator issue, in their experimental setting they only observe instability without warm-up.

We now aim to establish a predictive relationship between the **stuck-in-the-past** scenario and loss spikes. We present initial results in Figure 6.4, where we examine RMS_t for the the visual transformer

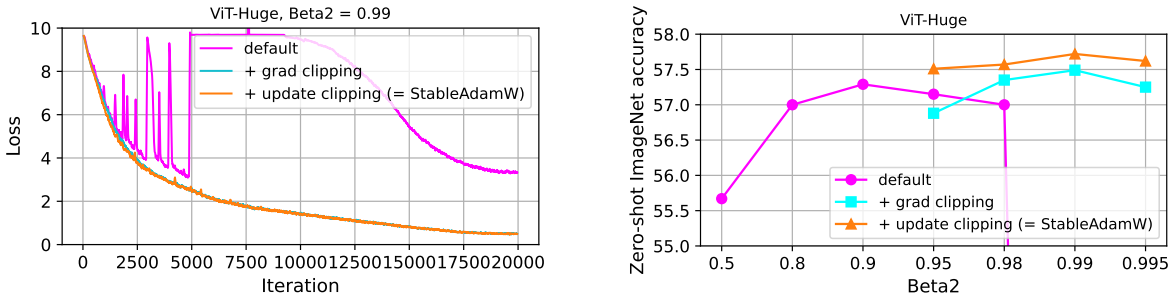

Figure 6.5: Adding update clipping to AdamW mitigates loss spikes and outperforms other interventions such as gradient clipping with norm 1. Code for the AdamW-AdaFactor hybrid we recommend of AdamW + update clipping is in Algorithm 6.2. The left plot shows loss curves for $\beta_2 = 0.99$ while the right displays accuracy ablating over β_2.

patch embedding layer, visual.conv1.weight. This means that the expectation is computed over parameters in visual.conv1.weight only. This figure illustrates a few important findings: i) loss spikes tend to follow 1-8 iterations after an RMS spike, ii) loss spikes slow learning as recovery time is required, and iii), RMS_t stays around 1 for lower β_2.

As this is just one example, we further elaborate on the predictive relationship between an RMS spike in the embedding layer in the Appendix. For analysis purposes, we define a heuristic to characterize loss and RMS spikes in visual.conv1.weight. We then show that 28 out of 30 detected loss spikes follow an RMS spike by 1-8 iterations, while the probability that a loss spike follows an RMS spike by chance is only 1%. Moreover, we find that the same predictive relationship does not exist for the RMS in other transformer layers.

6.4.5 StableAdamW: AdamW with update clipping from AdaFactor

This Section develops and tests StableAdamW (Algorithm 6.2), an AdamW-Adafactor hybrid.

To stabilize training, the AdaFactor optimizer divides the learning rate for iteration t by $1/\max(\text{RMS}_t, 1)$.[2] They refer to this as *update clipping*. The effect is to slow training when u_t is no longer a good estimator for g_t^2.

As discussed in Section 6.4.4, our stability issues can be traced to an out-of-date u_t which is what led Shazeer and Stern [234] to update clipping, even though their stability issues are also solved with warm-up. Therefore, we port update clipping to the standard AdamW optimizer with $d = 1$ and refer to the

[2]They actually introduce a hyperparameter d and use $1/\max(\text{RMS}_t/d, 1)$, but recommend setting $d = 1$.

resulting AdamW-Adafactor hybrid as StableAdamW (Algorithm 6.2). A modification we make is to compute and divide learning rate by $\max(\mathsf{RMS}_t, 1)$ independently for each tensor, which is for implementation convenience. This means that the expectation will be computed independently for each layer to produce a different RMS_t.

We now test how StableAdamW compares with other stability interventions such as gradient clipping[3] or lowering β_2. These results, presented in Figure 6.5 find that StableAdamW (i.e., AdamW + update clipping) outperforms these aforementioned interventions for CLIP ViT-Huge. While gradient clipping and update clipping both remove instability, update clipping performs better in terms of zero-shot ImageNet accuracy. With update or gradient clipping, higher β_2 such as 0.99 tends to perform better.

6.5 Limitations, broader impacts, and conclusion

We believe the main limitation of our work is that it is non-exhaustive. For instance, we only simulate float8 training and our experiments focus solely on CLIP-style training. In terms of broader impact, our work may enable additional CLIP models, whose broader impact is examined extensively by Section 7 of Radford et al. [210]. Finally, we believe that our findings on accelerating and stabilizing large multi-modal model training will be broadly useful to the community.

[3] We clip at global norm 1. We observed instability when trying 2 instead of 1. We did not tune this further, but note that 1.0 is standard in, e.g., PaLM [44], and Scaling Vision Transformers [292].

Chapter 7

Small-scale proxies for large-scale Transformer training instabilities

7.1 Overview

Teams that have trained large Transformer-based models have reported training instabilities at large scale that did not appear when training with the same hyperparameters at smaller scales. Although the causes of such instabilities are of scientific interest, the amount of resources required to reproduce them has made investigation difficult. In this work, we seek ways to reproduce and study training instability at smaller scales. First, we focus on two sources of training instability described in previous work: the growth of logits in attention layers [58] and divergence of the output logits from the log probabilities [44]. By measuring the relationship between learning rate and loss across scales, we show that these instabilities also appear in small models when training at high learning rates, and that mitigations previously employed at large scales are equally effective in this regime. This prompts us to investigate the extent to which other known optimizer and model interventions influence the sensitivity of the final loss to changes in the learning rate. To this end, we study methods such as warm-up, weight decay, and the μParam [283], and combine techniques to train small models that achieve similar losses across orders of magnitude of learning rate variation. Finally, to conclude our exploration we study two cases where instabilities can be predicted before they emerge by examining the scaling behavior of model activation and gradient norms.

7.2 Introduction

Scaling up transformers has led to remarkable progress from chat models to image generation. However, not every training run is successful. When training large Transformers, researchers have reported instabilities which slow or destabilize learning [44, 58, 301, 187, 49]. As the resources required for large runs continue to grow, it is important to examine the ways that Transformer training can fail.

In this report we reproduce, study, and predict training instability in Transformer models. We find that measuring the relationship between learning rate and loss across scales is a useful tool to identify instability (e.g., Figure 7.1). Therefore, we introduce learning rate (LR) sensitivity, which serves as a useful summary statistic for learning rate vs. loss curves. LR sensitivity measures the deviation from optimal performance when varying LR across orders of magnitude.

We show that two sources of instability, which have previously been described at scale, can be reproduced in small Transformers.[1] This enables their study without access to large resource pools. In particular, we examine the growth of logits in attention layers [58, 94, 291] and divergence of the output logits from the log probabilities [44]. As evident from the learning rate vs. loss curves and by inspecting model characteristics, both instabilities appear at high learning rates in small models. Moreover, interventions which have previously been employed at scale are also successful in this regime (e.g., Figure 7.1). These interventions—qk-layernorm [58] and z-loss regularization [44]—reduce LR sensitivity and enable successful training across three orders of magnitude of learning rate variation.

These observations raise the question of how other known optimizer and model interventions affect the shape of the learning rate vs. loss curves across scales. Therefore, we study the effect of techniques such as warm-up, weight decay, and μParam [283] in this context. When employing qk-layernorm and z-loss regularization, these other techniques usually have little impact on the range of learning rates at which models can be stably trained, but do affect the sensitivity to learning rate within this range. In line with previous work, we find that longer warm-up reduces learning rate sensitivity, as does the independent scaling of learning rate and weight decay recommended by Loshchilov and Hutter [169]. One interesting finding is that scaling depth increases LR sensitivity at a faster rate than scaling width.

The remainder of our investigation centers on the scaling behavior for model characteristics such as

[1] We focus on instabilities which lead to slow divergence, not loss spikes (see Section 7.5).

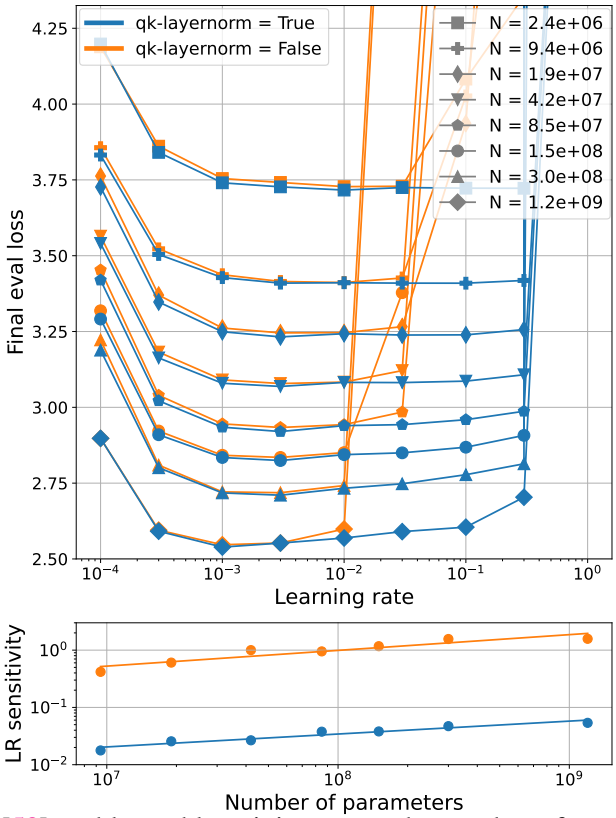

Figure 7.1: Qk-layernorm [58] enables stable training across three orders of magnitude of learning rate (LR) variation. **(Top)** For transformers with N parameters, we plot the effect of learning rate on final evaluation loss. **(Bottom)** We use LR sensitivity to summarize the top plot. LR sensitivity measures the expected deviation from the minimum achieved loss when varying learning rate across three orders of magnitude. Qk-layernorm reduces LR sensitivity, but LR sensitivity still increases with model scale.

activation and gradient norms. Using the attention logit growth instability as an example, we show that it is possible to predict an instability before it emerges. This is in contrast to prior works on scaling which primarily focus on scaling trends related to loss [132, 115].

We conclude by using the scaling behavior of model characteristics to search for instabilities that are currently not well documented. Our investigation shows that gradient norms decrease with both scale and learning rate, such that the default AdamW [169] epsilon hyperparameter is too large. This causes updates that are too small. We connect this phenomenon and the attention logit growth instability to parameter norm growth [182, 154].

Overall, we believe our work presents new scientific opportunities for studying training stability without access to large resource pools.

7.3 Experimental methodology

This section details our experimental set-up (Section 7.3.1) and useful tools employed by our analysis: (i) measuring the relationship between learning rate and loss across scales (Section 7.3.2) and (ii) examining scaling trends for model characteristics (Section 7.3.3).

7.3.1 Experimental set-up

We train small Transformer models [257] with a similar experimental set-up as GPT-2 [209] implemented in Flax [108]: the models are decoder-only [167] and trained with an auto-regressive loss. While we experimentally manipulate many of the following hyperparameters, this section provides their default values, which we use unless otherwise specified.

By default, we use AdamW [169] with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon =$ 1e-8, and gradient clipping at global norm 1. The default warmup is 5e3 steps, and the default number of total steps is 1e5. We use a linear schedule for warmup and and a cosine-decay [168] schedule for the remainder, with minimum learning rate 1e-5. We use an independent weight decay of 1e-4 and auxiliary z-loss [44] with coefficient 1e-4. Sections 7.4.2 and 7.4.1 respectively provide additional information and ablations on decoupled weight decay and z-loss. We use pre-normalization [209] Transformers with qk-layernorm [58] (see Section 7.4.1 for information). We do not use any biases following Chowdhery et al. [44], and the layernorm [11] ϵ

remains at the default value in Flax [108] of 1e-6. We jointly scale up the embedding size, depth, and number of heads when scaling parameters. We do not use weight tying of the first and last layer [206], and when reporting the number of parameters we exclude the embedding and head (as in Kaplan et al. [132]). We use rotary positional embeddings [244], and for training data we use C4 [214]. Letting d refer to the model dimension (i.e., the embedding size), the feed-forward component of the Transformer is an MLP with hidden dimension of $4d$ and gelu [111] activations. As in Vaswani et al. [257] we use factor $1/\sqrt{d}$ scaling in the self-attention. The embedding initialization is the default in Flax, which is normally distributed with standard deviation $1/\sqrt{d}$. The remainder of the weights are initialized with a truncated normal distribution with inverse root fan-in standard deviation [98]. The default batch size is 256, where each batch element has a sequence length of 512 tokens. Sequences are packed so that no padding is required. Finally, we use the vocabulary from Raffel et al. [215] which has size 32101 and uses a SentencePiece [145] tokenizer. We train on TPUs [128] in bfloat16 precision using Flax [108] and JAX [30].

7.3.2 LR vs. loss curves and learning rate sensitivity

To investigate how model instability emerges with scale, it is useful to plot the relationship between learning rate (LR) and loss for models of different sizes. For instance, an instability is often characterized by an explosion in the loss at high learning rates. LR vs. loss curves can reveal how the lowest unstable learning rate changes as a function of model size.

To summarize LR vs. loss curves, we use LR sensitivity. LR sensitivity measures the deviation in final validation loss from optimal when sweeping LR across three orders of magnitude. If a model fails to train at high learning rates, then LR sensitivity will be high. There are cases where LR vs. loss curves and LR sensitivity are no longer meaningful, for instance if an intervention changes the meaning of learning rate.

Let $\theta = \mathcal{A}(\eta)$ denote the model weights θ obtained when training with learning rate η, and let $\ell(\theta)$ denote the validation loss when using weights θ. For a learning rate range $[a, b]$, let ℓ^* denote the loss obtained with the best learning rate, i.e., $\ell^* = \min_{\eta \in [a,b]} \ell\left(\mathcal{A}(\eta)\right)$. Moreover, let ℓ_0 denote loss at initialization. Then, LR sensitivity is defined as $\mathbb{E}_{\eta \in [a,b]}\left[\min\left(\ell\left(\mathcal{A}\left(\eta\right)\right), \ell_0\right) - \ell^*\right]$.

Unless otherwise mentioned, we use the learning rate range 3e-4 to 3e-1 with AdamW [169] to measure LR sensitivity, where LR refers to the maximum value in a cosine decay schedule with warm-up [168]. We consider LRs in {3e-4, 1e-3, 3e-3, 1e-2, 3e-2, 1e-1, 3e-1} when computing the minimum and expectation.

7.3.3 Scaling trends for model characteristics

To study instability, we also find it useful to examine scaling trends for model characteristics such as gradient or activation norms. This method is helpful for predicting instabilities and contrasts with previous work on scaling, which primarily focuses on trends relating model scale and loss [132, 115].

7.4 Results

This section presents our results on training stability for small Transformers. Equipped with LR sensitivity (Section 7.3.2), we study two known instabilities and their corresponding mitigation at small scale (Section 7.4.1). This raises the question of how other model and optimizer interventions effect sensitivity of final loss to learning rate, which we investigate in Section 7.4.2. Finally, we examine whether instabilities can be reliably predicted before they emerge: Section 7.4.3 predicts when the logit growth instability may cause divergence in a larger model, while Section 7.4.4 aims to find other issues that may occur when scaling up with our default hyperparameters.

7.4.1 Reproducing two known instabilities at small scale

Here, we examine two instabilities that have previously been described at scale: the growth of logits in attention layers [58, 94, 291] and divergence of the output logits from the log probabilities [44]. By examining LR vs. loss curves, we show that these instabilities can be reproduced in small models by using high learning rates and that mitigations employed at scale are effective in this regime.

Attention logit growth

Researchers have previously documented that Transformer training fails when the attention logits become large [58, 291]. In Dehghani et al. [58], this issue emerged when training a ViT model [71] with 22 billion parameters.

In the self-attention layer of a Transformer [257], queries q_i and keys k_i are combined to compute the attention logits $z_{ij} = \langle q_i, k_j \rangle / \sqrt{d_h}$, where d_h is the head dimension. Next, the attention logits are passed through a softmax to produce attention weights, which are used to combine values v_i. Dehghani et al. [58] observed that the attention logits z became large, which they refered to as attention logit growth. As a result,

the attention weights collapse to one-hot vectors, which was named attention entropy collapse by Zhai et al. [291]. To resolve this issue, Dehghani et al. [58] proposed qk-layernorm, which applies LayerNorm [11] to the queries and keys before computing the attention logits.

In our experiments, we find that models need not be large to exhibit instability related to attention logit growth. As shown in Figure 7.1, the maximum learning rate at which small models can be trained increases when using qk-layernorm. Without qk-layernorm, the learning rate at which models diverge becomes smaller with increasing model size. By contrast, models with qk-layernorm exhibit considerably lower LR sensitivity and train to low loss at high learning rates. As a highlight, qk-layernorm allows training a model with 1.2B parameters at learning rate 0.3. Both with and without qk-layernorm, LR sensitivity increases with scale.

Output logit divergence

Another instability reported by researchers training large models is divergence in the output logits from the log probabilities [44]. Just as before, we reproduce this instability with small models at large learning rates, and the proposed mitigation ameliorates the issue. Overall, Figure 7.2 summarizes the effect.

Let y denote the model's output logits, which are used to compute class probabilities p_i via a softmax $p_i = e^{y_i}/Z$ where $Z = \sum_j e^{y_j}$. This instability occurs when the logits diverge and become very negative for a 2.4M parameter model at learning rate 0.1. In contrast to the attention logit growth instability, this divergence occurs towards the end of training. The mitigation proposed by Chowdhery et al. [44] is to encourage $\log Z$ to remain close to zero. They add an auxiliary loss $\log^2 Z$, referred to as z-loss, with coefficient 1e-4.

As illustrated in Figures 7.2, we find that instability related to output logit divergence occurs in models with no weight decay regardless of scale, and z-loss resolves this instability. Weight decay also mitigates this instability for the larger models we test.

7.4.2 Measuring the effect of other known interventions

The previous section used the relationship between learning rate and loss as a useful tool for examining two known instabilities and their mitigation. This raises the question of how other known model and optimizer

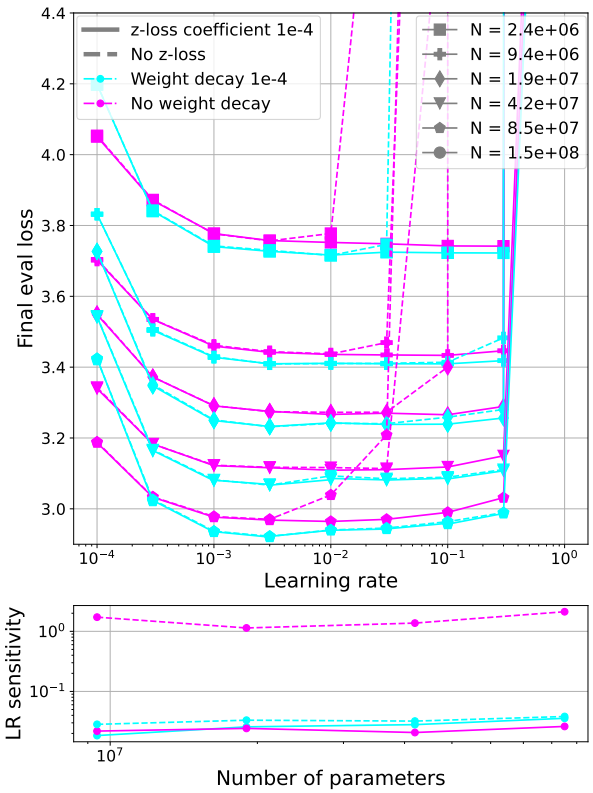

Figure 7.2: The effect of the output logit divergence instability [44] and the z-loss mitigation [44] (Section 7.4.1). Models in this experiment have qk-layernorm [58].

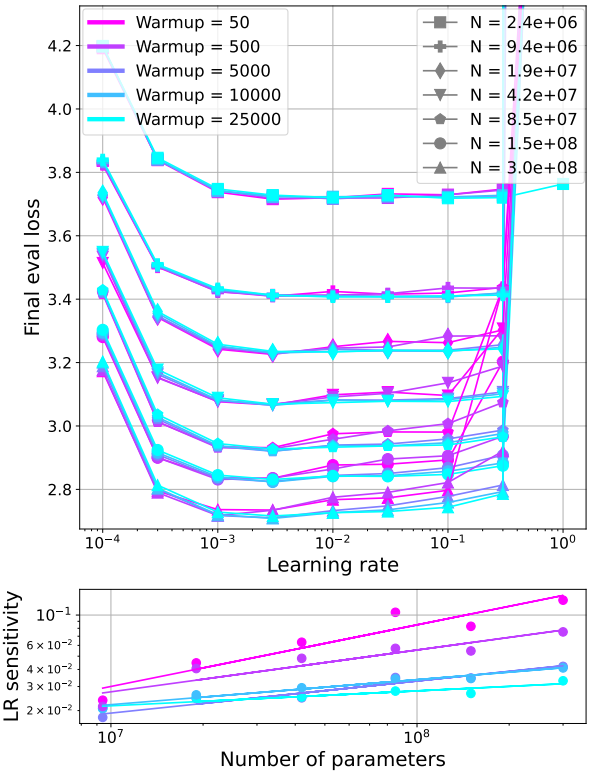

Figure 7.3: The effect of warm-up length for different model sizes. Longer warm-up reduces LR sensitivity and loss, especially for the larger models we test. Models in this experiment use qk-layernorm [58].

interventions affect the shape of LR vs. loss curves across scales. In particular, can LR sensitivity help identify additional issues or resolutions when scaling? This section aims to answer this question for common techniques such as warm-up, weight decay, and μParam [283].

Warm-up

As illustrated by Figure 7.3, a longer warm-up period reduces LR sensitivity. This is most clear for the larger models, which are not stable at LR 3e-1 without long warm-up. The number of total steps is fixed to 1e5 in this experiment, and all models use qk-layernorm. The importance of warm-up for stability has previously been highlighted [96, 234, 166], although these works do not measure scaling behavior.

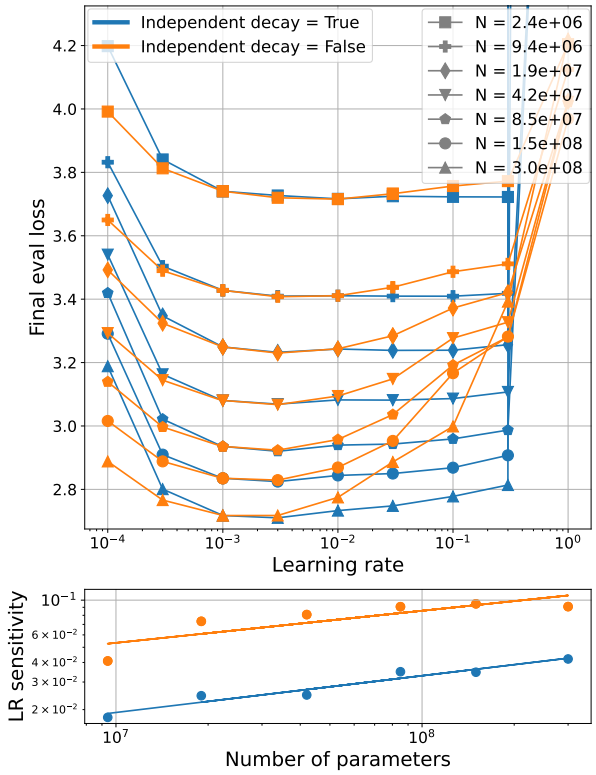

Figure 7.4: Independently scaling LR without also scaling weight decay reduces LR sensitivity. While this was recommended by Loshchilov and Hutter [169], it is not common practice in the default AdamW implementations in popular libraries. Refer to Section 7.4.2 for more detail.

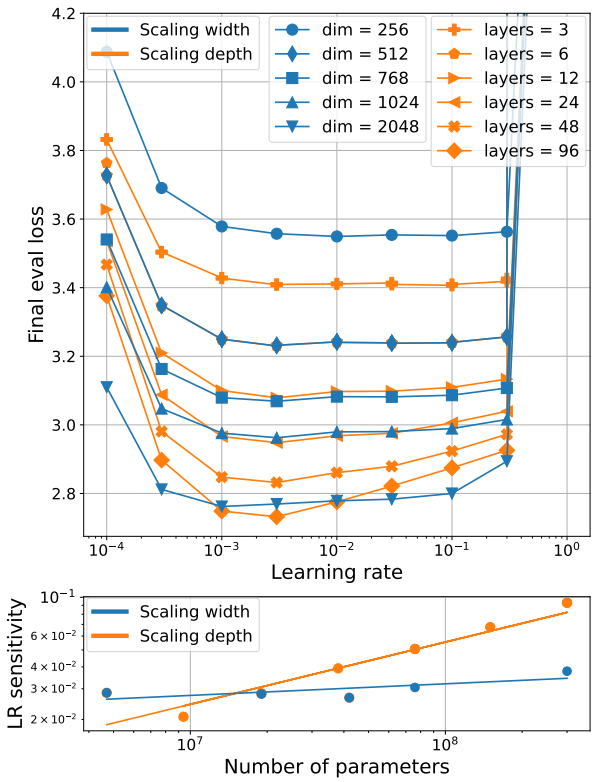

Figure 7.5: Independently scaling depth increases LR sensitivity at a faster rate than scaling width, though also produces a model with lower loss at the largest scale we test.

Independent weight decay

Parameterizing weight decay independently of learning rate reduces LR sensitivity, as illustrated in Figure 7.4. While this was recommended by Loshchilov and Hutter [169], it is not common practice in the default AdamW implementations of PyTorch [200] or Optax [12]. We explain the differences below.

For parameters θ, let $\Delta = v/\left(\sqrt{u} + \epsilon\right)$ denote the AdamW update without learning rate or weight decay. For weight decay coefficient λ, max learning rate η, and schedule $s_t \in [0, 1]$, Loshchilov and Hutter [169] recommend the update $\theta \leftarrow \theta - s_t(\eta\Delta - \lambda\theta)$, which we refer to as *independent decay*. On the other hand, the default implementation in PyTorch or Optax applies the update $\theta \leftarrow \theta - s_t\eta(\Delta - \lambda\theta)$, i.e., η now scales both terms.

When reporting LR sensitivity without independent decay in Figure 7.4, we report the minimum LR

sensitivity over ranges [1e-4, 1e-1] and [3e-4, 3e-1] because the former is sometimes better centered on the minimum. The default setting in this paper is to use independent decay. When using independent decay we set λ=1e-4, and without independent decay we set λ=0.1.

Scaling width vs. depth

We have so far consistently observed that increasing the number of parameters increases LR sensitivity. We now examine which part of scaling is most responsible.

Our results, illustrated by Figure 7.5, indicate that scaling depth increases LR sensitivity at a faster rate than scaling width. However, at the largest scale we test, independently scaling depth produces a model with lower validation loss. The standard practice of joint scaling performs best at the largest scale and also has a more reliable scaling prediction when extrapolating.

When scaling depth, we use $d = 512$, and when scaling width, we use 6 layers. The number of heads is scaled proportionally with width, so that the head dimension remains the same.

μParam

Yang and Hu [282] introduced the μParam method for parameterizing a neural network. As a product, the optimal LR remains consistent when scaling model width [283]. This section tests the effect of μParam on LR sensitivity, and examines whether μParam alleviates the need for qk-layernorm [58].

μParam does succeed in stabilizing the optimal LR at the scale we test. However, μParam does not improve loss or reduce LR sensitivity in our experiments. Our results indicate that μParam does not alleviate the need for this intervention at high learning rates. We note that from a practical perspective, reducing LR sensitivity is not important if the optimal LR does not change.

We refer to the variant of μParam that we use in these experiments as μParam (simple) because it maintains only the core feature of μParam. We add additional features from Yang et al. [283] without measurable improvement at the largest scale we test. For μParam (simple) we make the following changes from our standard baseline: scale the LR for linear layers by base-fan-in/fan-in. For μParam (full) there are three additional changes: (i) initialize the head with standard deviation $\sqrt{\text{base-fan-in}}/\text{fan-in}$; (ii) change the $1/\sqrt{d_h}$ scaling factor in attention layers to $1/d_h$ where d_h is the head dimension; and (iii) initialize the

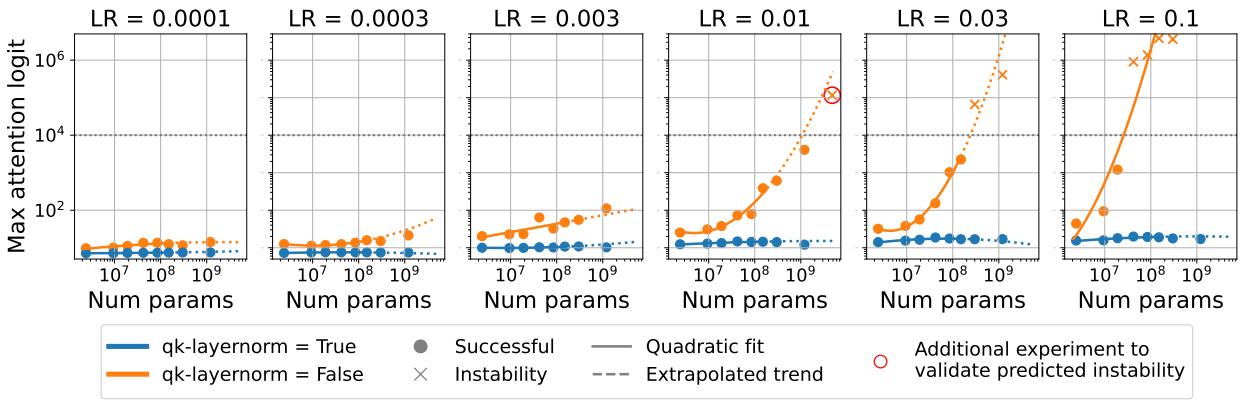

Figure 7.6: Predicting the attention logit growth instability via scaling behavior of model characteristics. We extrapolate to predict that a larger model will become unstable at LR 1e-2, and run an experiment to confirm the prediction. Refer to Section 7.4.3 for more information.

query projection weights with zeros. For base-fan-in we use the fan-in values for the smallest model we test, which has width 256. We ablate on change (ii) in isolation. In initial experiments (iii) had no noticeable effect.

7.4.3 Predicting attention logit growth instability from scaling behavior of model characteristics

A central question when studying instabilities is whether they can be predicted using small-scale proxy experiments. We now examine whether it is possible to predict the logit growth instability before it occurs using previous runs with smaller models. We track the attention logit maximums across model scales and fit a curve to the data. We use this to predict that a 4.8B parameter model will be unstable at LR 1e-2 without qk-layernorm and run an experiment to confirm this prediction.

Figure 7.6 plots the number of parameters vs. max attention logit at different learning rate values.[2] At each LR, we fit a quadratic to predict how the max attention logit will change with model scale.

We first noticed that all points with attention logits above 1e4 diverged. Moreover, the quadratic fit predicted that for LR 1e-2 the next model scale would also cross that value. Based on this prediction, we trained a new 4.8B parameter model at LR 1e-2. This model diverged as predicted. Not only do we predict the divergence, but our fit closely extrapolates to predict the value of max attention logit.

[2]We use block 0, which typically has the largest logits, and consider the value at step 2e3. Much earlier than 2e3 was uninformative, and much later the unstable points had long past diverged.

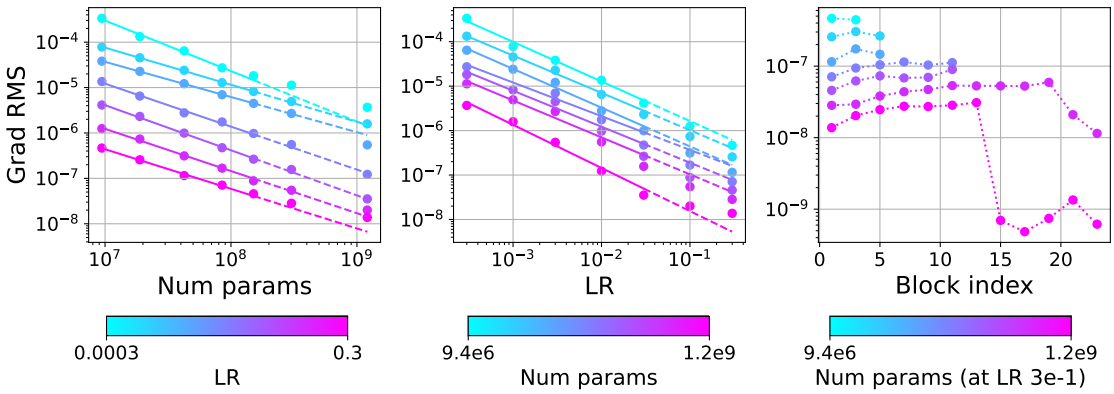

Figure 7.7: Predicting a potential instability from the scaling behavior of model characteristics. The gradient root mean square (RMS) decreases with num params (left) and learning rate (middle). These trends indicate that hyperparameter adjustment may be required to successfully scale further, as the RMS is approaching the default AdamW ϵ hyperparameter. If the gradient RMS becomes too small without adjusting ϵ or weight decay, a layer may collapse. The gradient RMS in the left and middle plot is reported for the first MLP layer of block 0, but we observe similar trends for other layers. Gradient RMS across different blocks is also reported (right). Gradient and update RMS are averaged over the final 500 steps.

7.4.4 Searching for new instabilities via scaling trends of model characteristics

This section examines whether the scaling behavior of model characteristics can be used to predict new issues with the default model and hyperparameter settings.

In Figure 7.7 we examine scaling trends for the gradient root mean square $\text{RMS}(g) = \sqrt{\mathbb{E}_i\left[g_i^2\right]}$. This figure reports the RMS for the first layer of the MLP, though we observe similar trends for other layers.

As models get larger, the value that grad RMS approaches is cause for concern. At the largest scale and learning rate we test, grad RMS is around the default AdamW ϵ hyperparameter. Recall that the unscaled AdamW update is $\Delta = v/\left(\sqrt{u}+\epsilon\right)$, where v and u are the first and second gradient moment EMA, respectively. If the grad RMS is on the same order as ϵ, then Δ will decrease in magnitude, and parameters will not receive learning signals as intended.

An obvious mitigation for this issue is to simply lower the AdamW ϵ hyperparameter from its default of 1e-8. Decreasing ϵ to 1e-15 improves loss and mitigates a collapse in grad RMS. We believe this improvement will only increase at scale. On the other hand, increasing ϵ to 1e-6 results in an instability.

Although we identified the instability above by empirically measuring the scaling behavior of the gradients, a mechanistic explanation exists. As learning rate increases, so does the parameter RMS. A larger parameter RMS leads to a larger RMS for the features output by each Transformer block. Then, the overall output RMS in turn increases with depth due to residual connections. The overall effect is that for larger

networks and learning rates, the Transformer output RMS entering the final layernorm will grow. Since the layernorm gradients are scaled by the inverse of their input RMS, the gradient received by the Transformer will shrink.

7.5 Related work

This paper mainly focuses on the effect of known interventions and instabilities, and so related work has been primarily discussed when relevant. This includes the attention growth instability observed by Dehghani et al. [58], Zhai et al. [291], and the final logit divergence issue encountered by Chowdhery et al. [44]. However, we highlight similar experimental methods in previous work. For instance, Yang et al. [283] also measure the relationship between LR and loss across scales, but their focus is on centering the optimum (see Section 7.4.2). In addition, Zhai et al. [291] elicit instability in base models by doubling learning rate, and Dettmers et al. [62] measure the presence of outlier features as a function of scale.

There are also important instabilities and related topics we have not directly discussed so far. For instance, we have primarily focused on instabilities that lead to a slow divergence, and we now summarize research on *fast loss spikes*. This instability is characterized by a quick increase in the loss that often eventually recovers.

The Edge of Stability and fast spikes. The conventional understanding of gradient descent predicts that loss instability only occurs when the learning rate exceeds $2/\lambda_{\text{max}}(H)$, where H is the Hessian. However recent investigations into large batch neural network training dynamics have revealed a more complicated picture via edge of stability (EoS) [48]. When training neural networks with large batch SGD, the loss curvature constantly evolves via the interaction of two processes: progressive sharpening and self stabilization. Progressive sharpening is the empirical observation that when LR $< 2/\lambda_{\text{max}}(H)$, the curvature gradually increases until the stability threshold is violated. When the learning rate becomes too large relative to the curvature, *fast loss spikes* occur and the parameters oscillate into a region with smaller $\lambda_{\text{max}}(H)$ where stable training and progressive sharpening resumes. The latter process where instability results in smaller $\lambda_{\text{max}}(H)$ is self-stabilization, a theoretical model of which is given in Damian et al. [54]. Gradually shrinking $\lambda_{\text{max}}(H)$ via self stabilization was shown to be a primary mechanism behind the success of learning rate warmup in Gilmer et al. [96], who closely studied the connections between curvature, initialization,

architecture and max trainable learning rates.

Cohen et al. [49] further analyze edge of stability of dynamics with adaptive optimizers, showing that progressive sharpening interacts with both the self-stabilization process and the adaptive optimizer state. This interaction results in the preconditioned sharpness $\lambda_{\text{max}}(P^{-1}H)$ oscillating around an optimizer specific threshold (38/LR in the case of Adam with β_1=0.9). Adaptive EoS (AEoS) can also result in periodic loss spikes when progressive sharpening pushes the preconditioned sharpness above the stability threshold, however the optimizer hyperparameters play a role. In particular, when LR$>$38/$\lambda_{\text{max}}(P^{-1}H)$, two mechanisms are now in play to resolve the step size being too big—either H can shrink or P^{-1} can shrink (or both). Cohen et al. [49] found that when β_2 is large, H tends to shrink and fast loss spikes result during the process, resembling the self stabilization process observed with gradient descent. However when β_2 is small, P^{-1} tends to shrink, no loss spikes are observed, and $\lambda_{\text{max}}(H)$ tends to gradually increase throughout training.

It is noteworthy that the adaptive edge of stability process (and the role of β_2) studied in Cohen et al. [49] offers a more complete understanding for loss spikes studied in a body of literature [234, 44, 187, 295, 37]. For example, Shazeer and Stern [234] argue that during training of Transformers with adaptive optimizers the optimizer update can become too big resulting in a loss spike followed by recovery. This is sometimes attributed to the adaptive optimizer state becoming "stale", which is consistent with the observation the reducing β_2 resolves the loss spikes [234, 295]. This is perhaps the same observation as Cohen et al. [49] that reducing β_2 allows P^{-1} to change quicker to adjust to the process of progressive sharpening. AEoS also offers an explanation for the periodic loss spikes observed when training large transformer models [187].

Parameter-free methods and more parameterizations. While our work has studied sensitivity to learning rate, there is also research that aims to eliminate the need to specify a learning rate [122, 57]. Based on their analysis, Ivgi et al. [122] set the step size for iteration t to the maximum distance from the initialization divided by the root sum of historical gradient squares. Moreover, while our work investigated μParam, there are additional parameterizations for which it would be interesting to explore LR vs. loss [67, 280, 27, 124].

7.6 Limitations and Conclusion

Limitations. We now describe some limitations with our work. First, we do not study the effect of data on instability. Second, we not make any explicit recommendations for practitioners on how to choose hyperparameters, as we believe this is beyond the scope of our investigation. Finally, a more systematic study would further substantiate the results of Section 7.4.3.

Conclusion. This paper demonstrates that useful insights on instability can be gained from small Transformers. Our results indicate that: (1) instabilities previously reported at scale can be reproduced in small-scale proxy models, facilitating their study without access to large resource pools; 2) instabilities previously reported at scale can be predicted before they emerge by extrapolating from experiments with small-scale proxy models; and 3) new instabilities can be found using small-scale proxy models.

Chapter 8

Replacing softmax with ReLU in Vision Transformers

8.1 Overview

Previous research observed accuracy degradation when replacing the attention softmax with a point-wise activation such as ReLU. In the context of vision transformers, we find that this degradation is mitigated when dividing by sequence length. Our experiments training small to large vision transformers on ImageNet-21k indicate that ReLU-attention can approach or match the performance of softmax-attention in terms of scaling behavior as a function of compute.

8.2 Introduction

The transformer architecture [257] is ubiquitous in modern machine learning. Attention, a central component of the transformer [14], includes a softmax which produces a probability distribution over tokens. Softmax is costly due to an exponent calculation and a sum over sequence length which makes parallelization challenging [208, 56].

In this report we explore point-wise alternatives to the softmax operation which do not necessarily output a probability distribution. As a highlight, we observe that attention with ReLU divided by sequence length can approach or match traditional softmax attention in terms of scaling behavior as a function of compute

for vision transformers. This result presents new opportunities for parallelization, as ReLU-attention can be parallelized over the sequence length dimension with fewer gather operations than traditional attention.

8.3 Related work

Previous research has explored substituting softmax with ReLU [235, 116] or squared ReLU [117]. However, these approaches do not divide by sequence length, which we experimentally find is important to reach accuracy comparable to softmax. In addition, previous research [164] has replaced softmax while still requiring normalization over the sequence length axis to ensure the attention weights sum to one. This retains the downside of requiring a gather. After writing an initial version of this note, it was brought to our attention that the variant of ReLU-atttention we study was also explored with a theoretical motivation [15, 86].

Moreover, there is extensive literature which removes activation functions altogether so that attention is linear [133, 170, 140], which is useful for long sequence lengths.[1] In our experiments, removing the activation entirely reduced accuracy.

8.4 Method

Attention. Attention transforms d-dimensional queries, keys, and values $\{q_i, k_i, v_i\}_{i=1}^L$ with a two step procedure. First, attention weights α_{ij} are produced via

$$\alpha_{ij} = \phi\left(\frac{1}{\sqrt{d}}\left[q_i^\top k_1, ..., q_i^\top k_L\right]\right)_j, \tag{8.1}$$

where ϕ is typically $\mathsf{softmax}$. Next, the attention weights are used to compute outputs $o_i = \sum_{j=1}^L \alpha_{ij} v_j$. This report explores point-wise alternatives to ϕ.

ReLU-attention. We observe that $\phi = L^{-1}\mathsf{relu}$ is a promising alternative to $\phi = \mathsf{softmax}$ in Equation 8.1. We refer to attention with $\phi = L^{-1}\mathsf{relu}$ as ReLU-attention.

Scaled point-wise attention. More generally, our experiments will explore $\phi = L^{-\alpha}h$ for $\alpha \in [0, 1]$ and $h \in \{\mathsf{relu}, \mathsf{relu}^2, \mathsf{gelu}, \mathsf{softplus}, \mathsf{identity}, \mathsf{relu6}, \mathsf{sigmoid}\}$ [52, 111].

[1] Concretely, with linear attention, the order of matrix multiplies can be switched from $(qk^\top)v$ to $q(k^\top v)$ which changes the compute required from $O(dL^2)$ to $O(d^2L)$ where $q, k, v \in \mathbb{R}^{L\times d}$ are the queries, keys, and values and L is sequence length.

Sequence length scaling. We observe that scaling by a term involving sequence length L is beneficial for high accuracy. This scaling is absent from prior work which removes softmax [117, 140]. While the central justification for sequence length scaling is empirical, we provide brief analytical motivation.

Transformers are currently designed with softmax attention for which $\sum_{j=1}^{L} \alpha_{ij} = 1$. This implies that $\mathbb{E}_j[\alpha_{ij}] = L^{-1}$. While it is unlikely that this is a necessary condition, $\phi = L^{-1}\text{relu}$ does ensure that $\mathbb{E}_j[\alpha_{ij}]$ is $O(L^{-1})$ at initialization. Preserving this condition may alleviate the need to change other hyperparameters when replacing softmax.

At initialization the elements of q and k are $O(1)$ and so $\frac{\langle q_i, k_j \rangle}{\sqrt{d}}$ will also be $O(1)$. Activation functions such as ReLU preserve $O(1)$,[2] and so a factor L^{-1} is necessary for $\mathbb{E}_j[\alpha_{ij}]$ to be $O(L^{-1})$.

8.5 Experiments

Experimental setup. Our experiments use ImageNet-21k and ImageNet-1k [59] training configurations from the BigVision codebase [23] without modifying hyperparameters.[3] In our experiments on ImageNet-21k we train for 30 epochs, and in our experiments on ImageNet-1k we train for 300 epochs. As a result, both training runs use a roughly similar number of steps of around 9e5. We use ViTs with qk-layernorm [58] as this was previously observed to be necessary to prevent instability when scaling model size. However, we ablate that this is not an important component at the scales we test. We use i21k and i1k to mean ImageNet-21k and ImageNet-1k, respectively, and report ImageNet-1k accuracy for ImageNet-21k models by taking the top class among those that are in ImageNet-1k, without fine-tuning. When evaluating transfer performance on downstream tasks we use a 10-shot linear probe averaged over three seeds. The downstream tasks are Caltech Birds [265], Caltech-101 [79], Stanford Cars [143], CIFAR-100 [144], DTD [46], ColHsit [134], Pets [199], and UC Merced [284].

Main experiment. Figure 8.1 illustrates that ReLU-attention matches the scaling trends for softmax attention for ImageNet-21k training. On the x-axis we display the total core hours required for the experiment. As an advantage, ReLU-attention enables parallelization over the sequence length dimension with

[2] With the exception of squared ReLU.

[3] For ImageNet1k we use the base config https://github.com/google-research/big_vision/blob/main/big_vision/configs/vit_i1k.py. For ImageNet21k we use the base config https://github.com/google-research/big_vision/blob/main/big_vision/configs/vit_i21k.py.

fewer gather operations than softmax attention.

Effect of sequence length scaling. Figure 8.2 examines the effect of sequence length scaling for various point-wise alternatives to softmax. Concretely, we replace softmax with $L^{-\alpha}h$ for $\alpha \in [0, 1]$ and $h \in \{\mathsf{relu}, \mathsf{relu}^2, \mathsf{gelu}, \mathsf{softplus}, \mathsf{identity}\}$. On the x-axis we display α. The y-axis displays accuracy for the S/32, S/16, and S/8 vision transformer models [71, 23]. The best results are typically achieved when α is close to 1. Since there is not clear best non-linearity, we use ReLU in our main experiment as it is faster.

Effect of qk-layernorm. Our main experiments use qk-layernorm [58] in which queries and keys are passed through LayerNorm [11] before computing attention weights. We use qk-layernorm by default as it was found to be necessary to prevent instability when scaling up model size [58]. Figure 8.3 shows the effect of removing qk-layernorm. The results indicate that qk-layernorm does not have a large effect for these models, but this may change at scale.

Effect of adding a gate. Previous work removing softmax adds a gated unit and does not scale by sequence length [117]. Concretely, in the gated attention unit [117] an extra projection produces output which is combined through elementwise-multiplication before the out projection. In Figure 8.4 we investigate whether the presence of a gate removes the need for sequence length scaling. Overall we observe that the best accuracy is still achieved with sequence length scaling, with or without the gate. Note that gating increases the core hours required for the experiment by roughly 9.3% for the S/8 model with ReLU.

8.6 Conclusion

This report leaves many open questions. In particular, we are unsure why the factor L^{-1} improves performance or if this term could be learned. Moreover, it is likely that there is a better activation function that we do not explore.

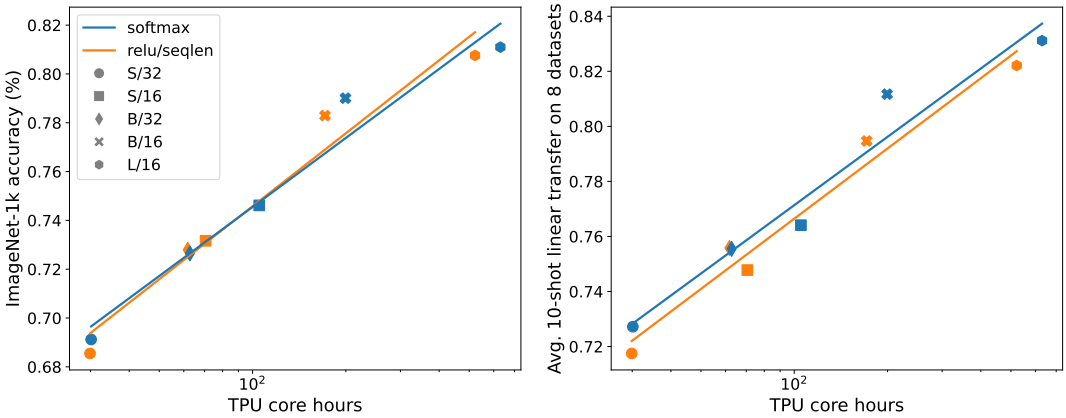

Figure 8.1: Replacing softmax with relu/seqlen approaches or matches the scaling performance of traditional attention for vision transformers [71] with qk-layernorm [58]. This figure displays results for small to large vision transformers trained on ImageNet-21k [59] for 30 epochs. We report ImageNet-1k accuracy for ImageNet-21k models by taking the top class among those that are in ImageNet-1k, without fine-tuning. Attention with ReLU can be parallelized over the sequence length dimension with less gather operations than softmax attention.

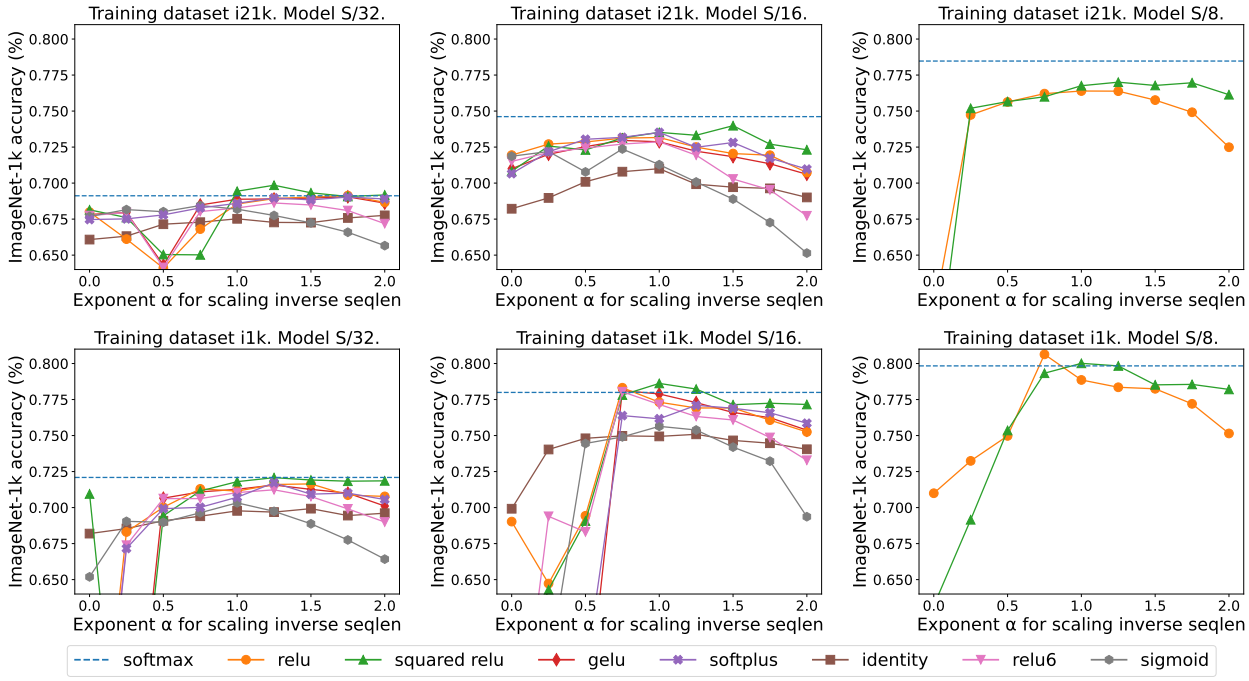

Figure 8.2: Replacing softmax with $L^{-\alpha}h$ where $h \in \{\text{relu}, \text{relu}^2, \text{gelu}, \text{softplus}, \text{identity}, \text{relu6}, \text{sigmoid}\}$ and L is sequence length. We typically observe the best results when α is close to 1. There is no clear best non-linearity at $\alpha \approx 1$, so we use ReLU in our main experiment for its speed.

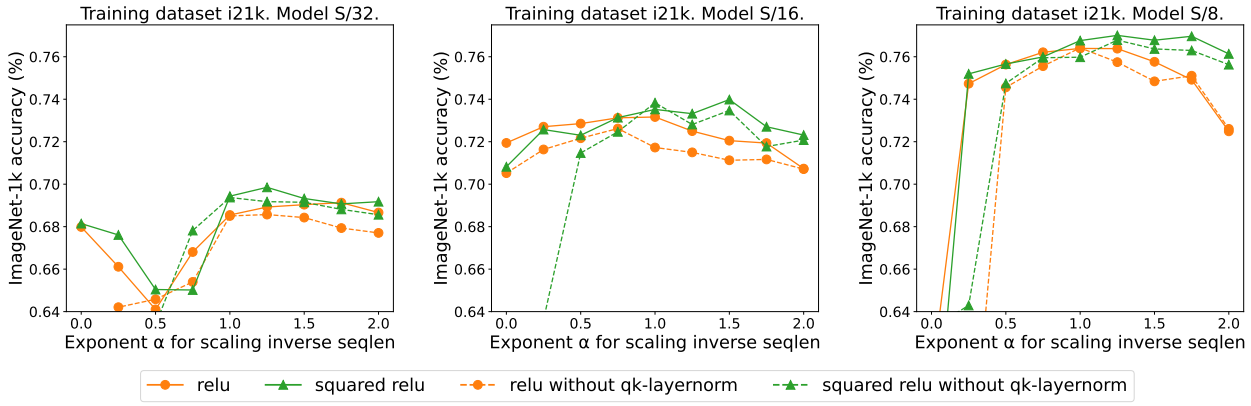

Figure 8.3: The effect of removing qk-layernorm [58] on attention with ReLU and squared ReLU scaled by $L^{-\alpha}$ where L is sequence length. Results are shown for the S/32, S/16, and S/8 vision transformer models [71, 23] trained on ImageNet-21k.

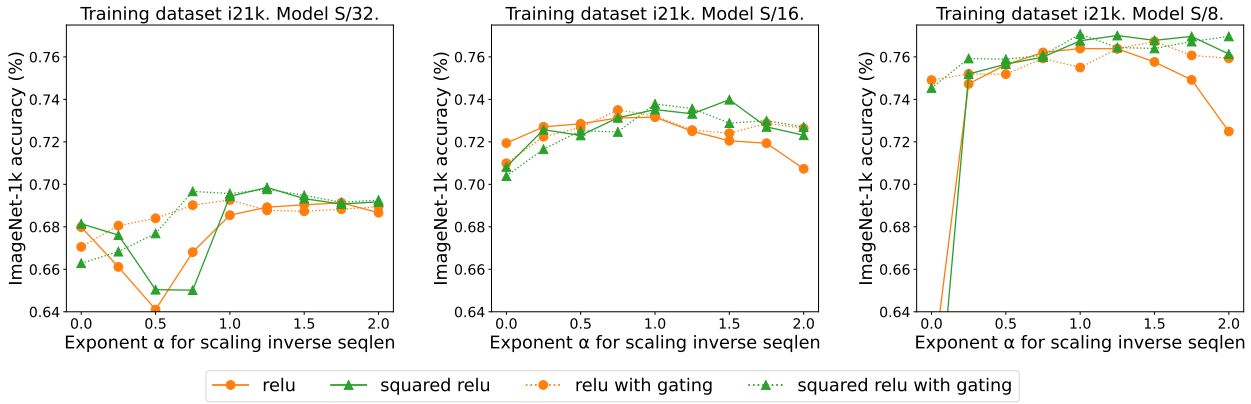

Figure 8.4: The effect of using a gated attention unit [117] on attention with ReLU and squared ReLU scaled by $L^{-\alpha}$ where L is sequence length. Results are shown for the S/32, S/16, and S/8 vision transformer models [71, 23] trained on ImageNet-21k.

Chapter 9

Conclusion

The past five years have witnessed marked progress in machine learning, from general chat systems and open-vocabulary image generators. Moreover, we believe that this progress is not slowing down. The current paradigm of scale is not yet exhausted, and we expect further advances in data, training algorithms, and models.

The research in this dissertation suggests two key directions for future development. First, the success of weight-interpolation (Chapters 2–5) suggests that advances in decentralized training are close-by. We expect research in this area to accelerate as pre-training scale increases, since it will be difficult for all accelerators to be co-located. In the future we believe that training will be asynchronous, employing a variant of local-SGD [242]. Douillard et al. [72] recently made promising steps towards this goal by building on our research in Chapters 3 and 5.

Next, our work on training instability uncovers new directions for model design. In particular, Chapter 8 demonstrates that Transformer self-attention can be replaced with a point-wise variant while preserving scaling trends and offering new opportunities for parallelism. There is still substantial work required in this direction. We suggest to replace softmax with a point-wise variant such as $\mathsf{ReLU}(\cdot)/\mathsf{seqlen}$, but it is currently not clear which non-linearity or scaling factor is optimal. Overall, we believe that advances in point-wise attention and analogous improvements will soon lead to a reliable model which eclipses the transformer.

Bibliography

[1] Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. Better fine-tuning by reducing representational collapse. In *International Conference on Learning Representations (ICLR)*, 2021. https://openreview.net/forum?id=OQ08SN70M1V.

[2] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. In *European conference on computer vision*, pages 329–344. Springer, 2014.

[3] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning, 2022. https://arxiv.org/abs/2204.14198.

[5] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. https://arxiv.org/abs/1811.11553.

[6] Anders Andreassen, Yasaman Bahri, Behnam Neyshabur, and Rebecca Roelofs. The evolution of out-of-distribution robustness throughout fine-tuning, 2021. https://arxiv.org/abs/2106.15831.

[7] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

[8] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015. https://arxiv.org/abs/1505.00468.

[9] Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, et al. Openflamingo: An open-source framework for training large autoregressive vision-language models. *arXiv preprint arXiv:2308.01390*, 2023.

[10] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 36–45, 2015.

[11] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[12] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

[13] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, 2021.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *arXiv preprint arXiv:2306.04637*, 2023.

[16] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising textual entailment challenge. In *Proc. of the II PASCAL challenge*, 2006.

[17] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper/2019/file/97af07a14cacba681feacf3012730892-Paper.pdf.

[18] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 1999. https://link.springer.com/article/10.1023/A:1007515423169.

[19] Sara Beery, Arushi Agarwal, Elijah Cole, and Vighnesh Birodkar. The iwildcam 2021 competition dataset. In *Conference on Computer Vision and Pattern Recognition (CVPR) FGVC8 Workshop*, 2021. https://arxiv.org/abs/2105.03494.

[20] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

[21] Gregory Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. In *International Conference on Machine Learning*, pages 769–779. PMLR, 2021.

[22] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[23] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv preprint arXiv:2205.01580*, 2022. URL https://arxiv.org/abs/2205.01580.

[24] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 2018. https://arxiv.org/abs/1712.03141.

[25] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 2013. https://arxiv.org/abs/1708.06131.

[26] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models, 2021. https://arxiv.org/abs/2108.07258.

[27] Blake Bordelon and Cengiz Pehlevan. Dynamics of finite width kernel and prediction fluctuations in mean field neural networks. *arXiv preprint arXiv:2304.03408*, 2023.

[28] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*, 2022.

[29] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European Conference on Computer Vision (ECCV)*, 2014. https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/.

[30] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[31] Leo Breiman. Bagging predictors. *Machine learning*, 1996. https://link.springer.com/article/10.1007/BF00058655.

[32] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR, 2021.

[33] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2005.14165.

[34] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[35] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. Getting the most out of ensemble selection. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 828–833. IEEE, 2006.

[36] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

[37] X. Chen, S. Xie, and K. He. An empirical study of training self-supervised vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9620–9629, Los Alamitos, CA, USA, oct 2021. IEEE Computer Society. doi: 10.1109/ICCV48922.2021.00950. URL https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.00950.

[38] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, et al. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

[39] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9640–9649, 2021.

[40] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark

and state of the art. *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, 2017. https://ieeexplore.ieee.org/abstract/document/7891544.

[41] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. *arXiv preprint arXiv:2212.07143*, 2022.

[42] Minsik Cho, Keivan A Vahid, Saurabh Adya, and Mohammad Rastegari. Dkm: Differentiable k-means clustering layer for neural network compression. *arXiv preprint arXiv:2108.12659*, 2021.

[43] Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. Fusing finetuned models for better pretraining. *arXiv preprint arXiv:2204.03044*, 2022.

[44] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[45] Gordon Christie, Neil Fendley, James Wilson, and Ryan Mukherjee. Functional map of the world. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. https://arxiv.org/abs/1711.07846.

[46] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. https://arxiv.org/abs/1311.3618.

[47] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. https://proceedings.mlr.press/v15/coates11a.html.

[48] Jeremy M Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065*, 2021.

[49] Jeremy M Cohen, Behrooz Ghorbani, Shankar Krishnan, Naman Agarwal, Sourabh Medapati, Michal

Badura, Daniel Suo, David Cardoze, Zachary Nado, George E Dahl, et al. Adaptive gradient methods at the edge of stability. *arXiv preprint arXiv:2207.14484*, 2022.

[50] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. RandAugment: Practical automated data augmentation with a reduced search space. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. https://arxiv.org/abs/1909.13719.

[51] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, 2005.

[52] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[53] Zihang Dai, Hanxiao Liu, Quoc Le, and Mingxing Tan. CoAtNet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34, 2021.

[54] Alex Damian, Eshaan Nichani, and Jason D Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. *arXiv preprint arXiv:2209.15594*, 2022.

[55] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning, 2020. https://arxiv.org/abs/2011.03395.

[56] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359, 2022.

[57] Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by d-adaptation. *arXiv preprint arXiv:2301.07733*, 2023.

[58] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer,

Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. *arXiv preprint arXiv:2302.05442*, 2023.

[59] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009. https://ieeexplore.ieee.org/document/5206848.

[60] Karan Desai and Justin Johnson. Virtex: Learning visual representations from textual annotations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. https://arxiv.org/abs/2006.06666.

[61] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*, 2022.

[62] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[63] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *9th International Conference on Learning Representations, ICLR*, 2022.

[64] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. URL https://aclanthology.org/N19-1423.

[65] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[66] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multi-*

ple classifier systems, 2000. https://link.springer.com/chapter/10.1007/3-540-45014-9_1.

[67] Emily Dinan, Sho Yaida, and Susan Zhang. Effective theory of transformers at initialization. *arXiv preprint arXiv:2304.02034*, 2023.

[68] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

[69] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proc. of IWP*, 2005.

[70] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[71] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021. https://arxiv.org/abs/2010.11929.

[72] Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

[73] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR, 2018.

[74] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[75] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations (ICLR)*, 2022. https://arxiv.org/abs/2110.06296.

[76] Christian Ertler, Jerneja Mislej, Tobias Ollmann, Lorenzo Porzi, Gerhard Neuhold, and Yubin Kuang. The mapillary traffic sign dataset for detection and classification on a global scale. In *European Conference on Computer Vision (ECCV)*, 2020. https://arxiv.org/abs/1909.04422.

[77] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. https://arxiv.org/abs/1707.08945.

[78] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

[79] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004.

[80] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=6Tm1mposlrM.

[81] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2010.15110.

[82] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning (ICML)*, 2020. https://arxiv.org/abs/1912.05671.

[83] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 1999. https://www.sciencedirect.com/science/article/pii/S1364661399012942.

[84] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 1997. https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[85] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*. Springer series in statistics New York, 2001.

[86] Hengyu Fu, Tianyu Guo, Yu Bai, and Song Mei. What can a single attention layer learn? a study through the random features lens. *arXiv preprint arXiv:2307.11353*, 2023.

[87] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *arXiv preprint arXiv:2304.14108*, 2023.

[88] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2021. URL https://arxiv.org/abs/2101.00027.

[89] Peng Gao, Shijie Geng, Renrui Zhang, Teli Ma, Rongyao Fang, Yongfeng Zhang, Hongsheng Li, and Yu Qiao. Clip-adapter: Better vision-language models with feature adapters. *arXiv preprint arXiv:2110.04544*, 2021.

[90] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. https://arxiv.org/abs/1802.10026.

[91] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. https://ieeexplore.ieee.org/abstract/document/6248074.

[92] Robert Geirhos, Carlos R Medina Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. https://arxiv.org/abs/1808.08750.

[93] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third pascal recognizing textual entailment challenge. In *Proc. of the ACL-PASCAL workshop on textual entailment and paraphrasing*, 2007.

[94] Justin Gilmer, Andrea Schioppa, and Jeremy Cohen. Intriguing properties of transformer training instabilities, . To appear.

[95] Justin Gilmer, Andrea Schioppa, and Jeremy Cohen. Intriguing properties of transformer training instabilities, . To appear.

[96] Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George Dahl, Zachary Nado, and Orhan Firat. A loss curvature perspective on training instability in deep learning. *arXiv preprint arXiv:2110.04369*, 2021.

[97] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[98] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[99] Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. https://distill.pub/2021/multimodal-neurons.

[100] Raphael Gontijo-Lopes, Yann Dauphin, and Ekin D. Cubuk. No one representation to rule them all: overlapping features of training methods, 2021. https://arxiv.org/abs/2110.12899.

[101] Raphael Gontijo-Lopes, Yann Dauphin, and Ekin Dogus Cubuk. No one representation to rule them all: Overlapping features of training methods. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=BK-4qbGgIE3.

[102] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations (ICLR)*, 2014. https://arxiv.org/abs/1412.6544.

[103] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, 2017. https://arxiv.org/abs/1706.04599.

[104] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

[105] Suchin Gururangan, Mitchell Wortsman, Samir Yitzhak Gadre, Achal Dave, Maciej Kilian, Weijia Shi, Jean Mercat, Georgios Smyrnis, Gabriel Ilharco, Matt Jordan, Reinhard Heckel, Alex Dimakis, Ali Farhadi, Vaishaal Shankar, and Ludwig Schmidt. Introducing openlm. https://laion.ai/blog/open-lm/, 2023.

[106] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. https://arxiv.org/abs/1512.03385.

[108] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

[109] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019. https://arxiv.org/abs/1709.00029.

[110] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *International Conference on Learning Representations (ICLR)*, 2019. https://arxiv.org/abs/1903.12261.

[111] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[112] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *International Conference on Computer Vision (ICCV)*, 2021. https://arxiv.org/abs/2006.16241.

[113] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. https://arxiv.org/abs/1907.07174.

[114] John Hewitt, Xiang Lisa Li, Sang Michael Xie, Benjamin Newman, and Percy Liang. Ensembles and cocktails: Robust finetuning for natural language generation. In *NeurIPS 2021 Workshop on Distribution Shifts*, 2021. https://openreview.net/forum?id=qXucB21w1C3.

[115] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[116] Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: Nngp and ntk for deep attention networks. In *International Conference on Machine Learning*, pages 4376–4386. PMLR, 2020.

[117] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International Conference on Machine Learning*, pages 9099–9117. PMLR, 2022.

[118] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[119] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[120] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021. URL https://doi.org/10.5281/zenodo.5143773. If you use this software, please cite it as below.

[121] Gabriel Ilharco, Mitchell Wortsma, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights, 2022. https://arxiv.org/abs/2208.05592.

[122] Maor Ivgi, Oliver Hinder, and Yair Carmon. Dog is sgd's best friend: A parameter-free dynamic step size schedule. *arXiv preprint arXiv:2302.12022*, 2023.

[123] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018. https://arxiv.org/abs/1803.05407.

[124] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. https://arxiv.org/abs/1806.07572.

[125] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V Le, Yunhsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2102.05918.

[126] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Association for Computational Linguistics (ACL)*, 2019. https://arxiv.org/abs/1911.03437.

[127] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. https://arxiv.org/abs/1612.06890.

[128] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[129] Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. The amu-uedin submission to the wmt16 news translation task: Attention-based nmt models as feature functions in phrase-based smt. *arXiv preprint arXiv:1605.04809*, 2016.

[130] Jean Kaddour, Linqing Liu, Ricardo Silva, and Matt J Kusner. Questions for flat-minima optimization of modern neural networks. *arXiv preprint arXiv:2202.00661*, 2022.

[131] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2): 1–210, 2021.

[132] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. https://arxiv.org/abs/2001.08361.

[133] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh,

editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

[134] Jakob Nikolas Kather, Frank Gerrit Zöllner, Francesco Bianconi, Susanne M Melchers, Lothar R Schad, Timo Gaiser, Alexander Marx, and Cleo-Aron Weis. Collection of textures in colorectal cancer histology. *Zenodo https://doi. org/10*, 5281, 2016.

[135] Daya Khudia, Jianyu Huang, Protonu Basu, Summer Deng, Haixin Liu, Jongsoo Park, and Mikhail Smelyanskiy. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

[136] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014. https://arxiv.org/abs/1412.6980.

[137] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences (PNAS)*, 2017. https://arxiv.org/abs/1612.00796.

[138] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2012.07421.

[139] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European Conference on Computer Vision (ECCV)*, 2020. https://arxiv.org/abs/1912.11370.

[140] Soroush Abbasi Koohpayegani and Hamed Pirsiavash. Sima: Simple softmax-free attention for vision transformers. *arXiv preprint arXiv:2206.08898*, 2022.

[141] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning (ICML)*, 2019. https://arxiv.org/abs/1905.00414.

[142] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. https://arxiv.org/abs/1805.08974.

[143] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *International Conference on Computer Vision (ICCV) Workshops*, 2013. https://ieeexplore.ieee.org/document/6755945.

[144] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[145] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[146] Ananya Kumar, Aditi Raghunathan, Tengyu Ma, and Percy Liang. Calibrated ensembles: A simple way to mitigate ID-OOD accuracy tradeoffs. In *NeurIPS 2021 Workshop on Distribution Shifts*, 2021. https://openreview.net/forum?id=dmDE-9e9F_x.

[147] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations (ICLR)*, 2022. https://openreview.net/forum?id=UYneFzXSJWh.

[148] Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=UYneFzXSJWh.

[149] Manoj Kumar, Mostafa Dehghani, and Neil Houlsby. Dual patchnorm. *arXiv preprint arXiv:2302.01327*, 2023.

[150] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 2003. https://doi.org/10.1023/A:1022859003006.

[151] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. https://arxiv.org/abs/1612.01474.

[152] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[153] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2010.

[154] Jaehoon Lee. A random walk model of transformer parameter growth, 2023.

[155] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. https://arxiv.org/abs/1703.08475.

[156] Julien-Charles Lévesque, Christian Gagné, and Robert Sabourin. Bayesian hyperparameter optimization for ensemble learning. *arXiv preprint arXiv:1605.06394*, 2016.

[157] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models, 2022. https://arxiv.org/abs/2206.14858.

[158] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[159] Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Rethinking the hyperparameters for fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2020. https://arxiv.org/abs/2002.11770.

[160] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

[161] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[162] Xingjian Li, Haoyi Xiong, Haozhe An, Cheng-Zhong Xu, and Dejing Dou. Rifle: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer. In *International Conference on Machine Learning*, pages 6010–6019. PMLR, 2020.

[163] Yanghao Li, Haoqi Fan, Ronghang Hu, Christoph Feichtenhofer, and Kaiming He. Scaling language-image pre-training via masking. *arXiv preprint arXiv:2212.00794*, 2022.

[164] Zhiyuan Li, Srinadh Bhojanapalli, Manzil Zaheer, Sashank Reddi, and Sanjiv Kumar. Robust training of neural networks using scale invariant architectures. In *International Conference on Machine Learning*, pages 12656–12684. PMLR, 2022.

[165] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtárik. Acceleration for compressed gradient descent in distributed and federated optimization. *arXiv preprint arXiv:2002.11364*, 2020.

[166] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[167] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hyg0vbWC-.

[168] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2016. https://arxiv.org/abs/1608.03983.

[169] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. https://openreview.net/forum?id=Bkg6RiCqY7.

[170] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. Soft: Softmax-free transformer with linear complexity. *Advances in Neural Information Processing Systems*, 34:21297–21309, 2021.

[171] Shangyun Lu, Bradley Nott, Aaron Olson, Alberto Todeschini, Hossein Vahabi, Yair Carmon, and Ludwig Schmidt. Harder or different? a closer look at distribution shift in dataset reproduction. In *International Conference on Machine Learning (ICML) Workshop on Uncertainty and Robustness in Deep Learning*, 2020. http://www.gatsby.ucl.ac.uk/~balaji/udl2020/accepted-papers/UDL2020-paper-101.pdf.

[172] Ekdeep Singh Lubana, Puja Trivedi, Danai Koutra, and Robert P. Dick. How do quadratic regularizers prevent catastrophic forgetting: The role of interpolation, 2021. https://arxiv.org/abs/2102.02805.

[173] James Lucas, Juhan Bae, Michael R Zhang, Stanislav Fort, Richard Zemel, and Roger Grosse. Analyzing monotonic linear interpolation in neural network loss landscapes. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2104.11044.

[174] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2017. https://arxiv.org/abs/1706.06083.

[175] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *European Conference on Computer Vision (ECCV)*, 2018. https://arxiv.org/abs/1805.00932.

[176] Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging, 2021. https://arxiv.org/abs/2111.09832.

[177] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 1975.

[178] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 1989. https://www.sciencedirect.com/science/article/pii/S0079742108605368.

[179] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*. Elsevier, 1989. https://www.sciencedirect.com/science/article/abs/pii/S0079742108605368.

[180] Ryan McDonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, volume 22, 2009.

[181] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65. PMLR, 2016.

[182] William Merrill, Vivek Ramanujan, Yoav Goldberg, Roy Schwartz, and Noah Smith. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. *arXiv preprint arXiv:2010.09697*, 2020.

[183] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[184] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

[185] John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning (ICML)*, 2020. https://arxiv.org/abs/2004.14444.

[186] John P Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2107.04649.

[187] Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, et al. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*, 2023.

[188] Norman Mu, Alexander Kirillov, David Wagner, and Saining Xie. Slip: Self-supervision meets language-image pre-training. *arXiv preprint arXiv:2112.12750*, 2021.

[189] Basil Mustafa, Carlos Riquelme, Joan Puigcerver, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Deep ensembles for low-data transfer learning, 2020. https://arxiv.org/abs/2010.06866.

[190] Vaishnavh Nagarajan and J. Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/05e97c207235d63ceb1db43c60db7bbb-Paper.pdf.

[191] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2011. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/37648.pdf.

[192] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2008.11687.

[193] John Nguyen, Kshitiz Malik, Maziar Sanjabi, and Michael Rabbat. Where to begin? exploring

the impact of pre-training and initialization in federated learning. *arXiv preprint arXiv:2206.15387*, 2022.

[194] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018. https://arxiv.org/abs/1803.02999.

[195] OpenAI. Gpt-4 technical report, 2023. URL https://ar5iv.org/abs/2303.08774.

[196] Jose Javier Gonzalez Ortiz, Jonathan Frankle, Mike Rabbat, Ari Morcos, and Nicolas Ballas. Trade-offs of local sgd at scale: An empirical study. *arXiv preprint arXiv:2110.08133*, 2021.

[197] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[198] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1906.02530.

[199] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[200] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1912.01703.

[201] Hieu Pham, Zihang Dai, Golnaz Ghiasi, Hanxiao Liu, Adams Wei Yu, Minh-Thang Luong, Mingxing Tan, and Quoc V Le. Combined scaling for zero-shot transfer learning, 2021. https://arxiv.org/abs/2111.10050.

[202] Hieu Pham, Zihang Dai, Golnaz Ghiasi, Hanxiao Liu, Adams Wei Yu, Minh-Thang Luong, Mingxing

Tan, and Quoc V. Le. Combined scaling for zero-shot transfer learning, 2021. https://arxiv.org/abs/2111.10050.

[203] Krishna Pillutla, Kshitiz Malik, Abdel-Rahman Mohamed, Mike Rabbat, Maziar Sanjabi, and Lin Xiao. Federated learning with partial model personalization. In *International Conference on Machine Learning*, pages 17716–17758. PMLR, 2022.

[204] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 1992. https://epubs.siam.org/doi/abs/10.1137/0330046?journalCode=sjcodc.

[205] Boris Teodorovich Polyak. New method of stochastic approximation type. *Automation and remote control*, 1990.

[206] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2025.

[207] Joaquin Quiñonero-Candela, Masashi Sugiyama, Neil D Lawrence, and Anton Schwaighofer. *Dataset shift in machine learning*. Mit Press, 2009.

[208] Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

[209] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners, 2019. https://openai.com/blog/better-language-models/.

[210] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2103.00020.

[211] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[212] Colin Raffel. A call to build models like we build open-source software, 2021. https://colinraffel.com/blog/a-call-to-build-models-like-we-build-open-source-software.html.

[213] Colin Raffel. A call to build models like we build open-source software, 2021. https://colinraffel.com/blog/a-call-to-build-models-like-we-build-open-source-software.html.

[214] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020. http://jmlr.org/papers/v21/20-074.html.

[215] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

[216] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11893–11902, 2020.

[217] Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations (ICLR)*, 2021. https://openreview.net/forum?id=GhVS8_yPeEa.

[218] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[219] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[220] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. https://arxiv.org/abs/1611.07725.

[221] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24, 2011.

[222] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning (ICML)*, 2019. https://arxiv.org/abs/1902.10811.

[223] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[224] Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of nlp models. In *Association for Computational Linguistics (ACL)*, 2022. https://www.microsoft.com/en-us/research/publication/adaptive-testing-and-debugging-of-nlp-models/.

[225] Rebecca Roelofs, Nicholas Cain, Jonathon Shlens, and Michael C Mozer. Mitigating bias in calibration error estimation, 2020. https://arxiv.org/abs/2012.08668.

[226] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[227] David Ruppert. Efficient estimations from a slowly convergent robbins-monro process, 1988. https://ecommons.cornell.edu/bitstream/handle/1813/8664/TR000781.pdf.

[228] Tonmoy Saikia, Thomas Brox, and Cordelia Schmid. Optimized generic feature learning for few-shot classification across domains. *arXiv preprint arXiv:2001.07926*, 2020.

[229] Mert Bulent Sariyildiz, Julien Perez, and Diane Larlus. Learning visual representations with caption annotations. In *European Conference on Computer Vision (ECCV)*, 2020. https://arxiv.org/abs/2008.01392.

[230] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *arXiv preprint arXiv:2210.08402*, 2022.

[231] Vaishaal Shankar, Achal Dave, Rebecca Roelofs, Deva Ramanan, Benjamin Recht, and Ludwig Schmidt. Do image classifiers generalize across time?, 2019. https://arxiv.org/abs/1906.02168.

[232] Vaishaal Shankar, Rebecca Roelofs, Horia Mania, Alex Fang, Benjamin Recht, and Ludwig Schmidt. Evaluating machine accuracy on imagenet. In *International Conference on Machine Learning (ICML)*, 2020. http://proceedings.mlr.press/v119/shankar20c/shankar20c.pdf.

[233] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014. https://arxiv.org/abs/1403.6382.

[234] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[235] Kai Shen, Junliang Guo, Xu Tan, Siliang Tang, Rui Wang, and Jiang Bian. A study on relu and softmax in transformer. *arXiv preprint arXiv:2302.06461*, 2023.

[236] Sheng Shen, Liunian Harold Li, Hao Tan, Mohit Bansal, Anna Rohrbach, Kai-Wei Chang, Zhewei Yao, and Kurt Keutzer. How much can clip benefit vision-and-language tasks? In *International Conference on Learning Representations (ICLR)*, 2022. https://arxiv.org/abs/2107.06383.

[237] Yang Shu, Zhi Kou, Zhangjie Cao, Jianmin Wang, and Mingsheng Long. Zoo-tuning: Adaptive transfer from a zoo of models. In *International Conference on Machine Learning*, pages 9626–9637. PMLR, 2021.

[238] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

[239] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, 2013.

[240] Shagun Sodhani, Olivier Delalleau, Mahmoud Assran, Koustuv Sinha, Nicolas Ballas, and Michael Rabbat. A closer look at codistillation for distributed training. *arXiv preprint arXiv:2010.02838*, 2020.

[241] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *International Joint Conference on Neural Networks (IJCNN)*, 2011. https://ieeexplore.ieee.org/document/6033395.

[242] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[243] Asa Cooper Stickland and Iain Murray. Diverse ensembles improve calibration. In *International Conference on Machine Learning (ICML) Workshop on Uncertainty and Robustness in Deep Learning*, 2020. https://arxiv.org/abs/2007.04206.

[244] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[245] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision (ICCV)*, 2017. https://arxiv.org/abs/1707.02968.

[246] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. https://arxiv.org/abs/1912.04838.

[247] Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. https://arxiv.org/abs/2111.09839.

[248] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. https://arxiv.org/abs/2112.06825.

[249] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[250] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2007.00644.

[251] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, 1998. https://link.springer.com/chapter/10.1007/978-1-4615-5529-2_8.

[252] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[253] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. https://people.csail.mit.edu/torralba/publications/datasets_cvpr11.pdf.

[254] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Go-

ing deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021.

[255] Hugo Touvron, Matthieu Cord, and Herve Jegou. Deit iii: Revenge of the vit. *arXiv preprint arXiv:2204.07118*, 2022.

[256] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*, 2017. https://arxiv.org/abs/1705.07204.

[257] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[258] Johannes Von Oswald, Seijin Kobayashi, Joao Sacramento, Alexander Meulemans, Christian Henning, and Benjamin F Grewe. Neural networks with late-phase weights. *arXiv preprint arXiv:2007.12927*, 2020.

[259] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[260] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1905.13549.

[261] Jue Wang, Binhang Yuan, Luka Rimanic, Yongjun He, Tri Dao, Beidi Chen, Christopher Re, and Ce Zhang. Fine-tuning language models over slow networks using activation compression with guarantees. *arXiv preprint arXiv:2206.01299*, 2022.

[262] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors,

Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/335d3d1cd7ef05ec77714a215134914c-Paper.pdf.

[263] Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. 2019. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus.

[264] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *TACL*, 7:625–641, 2019.

[265] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[266] Lilian Weng and Greg Brockman. Techniques for training large neural networks, 2022. https://openai.com/blog/techniques-for-training-large-neural-networks/.

[267] Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *arXiv preprint arXiv:2006.13570*, 2020.

[268] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

[269] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2006.14769.

[270] Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning (ICML)*, 2021. https://arxiv.org/abs/2102.10472.

[271] Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR, 2021.

[272] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. 2021. https://arxiv.org/abs/2109.01903.

[273] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

[274] Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.

[275] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. https://arxiv.org/abs/1708.07747.

[276] Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 2016. https://link.springer.com/article/10.1007/s11263-014-0748-y.

[277] Cong Xie, Shuai Zheng, Sanmi Koyejo, Indranil Gupta, Mu Li, and Haibin Lin. Cser: Communication-efficient sgd with error reset. *Advances in Neural Information Processing Systems*, 33:12593–12603, 2020.

[278] LI Xuhong, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *International Conference on Machine Learning*, pages 2825–2834. PMLR, 2018.

[279] Chhavi Yadav and Léon Bottou. Cold case: The lost mnist digits. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1905.10498.

[280] Sho Yaida. Meta-principled family of hyperparameter scaling strategies. *arXiv preprint arXiv:2210.04909*, 2022.

[281] I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification, 2019. https://arxiv.org/abs/1905.00546.

[282] Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pages 11727–11737. PMLR, 2021.

[283] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[284] Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pages 270–279, 2010.

[285] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. https://arxiv.org/abs/1411.1792.

[286] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[287] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu.

Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917*, 2022.

[288] Binhang Yuan, Yongjun He, Jared Quincy Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. *arXiv preprint arXiv:2206.01288*, 2022.

[289] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[290] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning (ICML)*, 2017. https://arxiv.org/abs/1703.04200.

[291] Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Josh Susskind. Stabilizing transformer training by preventing attention entropy collapse. *arXiv preprint arXiv:2303.06296*, 2023.

[292] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers, 2021. https://arxiv.org/abs/2106.04560.

[293] Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. Lit: Zero-shot transfer with locked-image text tuning, 2021. https://arxiv.org/abs/2111.07991.

[294] Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. Lit: Zero-shot transfer with locked-image text tuning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. https://arxiv.org/abs/2111.07991.

[295] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. *arXiv preprint arXiv:2303.15343*, 2023.

[296] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. 2017. https://arxiv.org/abs/1710.09412.

[297] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[298] Michael R Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. https://arxiv.org/abs/1907.08610.

[299] Renrui Zhang, Rongyao Fang, Peng Gao, Wei Zhang, Kunchang Li, Jifeng Dai, Yu Qiao, and Hongsheng Li. Tip-adapter: Training-free clip-adapter for better vision-language modeling. *arXiv preprint arXiv:2111.03930*, 2021.

[300] Susan Zhang. Open pretrained transformers lecture, 2023. https://www.youtube.com/watch?v=p9IxoSkvZ-M.

[301] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[302] Yuhao Zhang, Hang Jiang, Yasuhide Miura, Christopher D Manning, and Curtis P Langlotz. Contrastive learning of medical visual representations from paired images and text, 2020. https://arxiv.org/abs/2010.00747.

[303] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models, 2021. https://arxiv.org/abs/2109.01134.

[304] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Conditional prompt learning for vision-language models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. https://arxiv.org/abs/2203.05557.

[305] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced

adversarial training for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2020. https://arxiv.org/abs/1909.11764.

[306] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. Towards unified int8 training for convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1969–1979, 2020.

[307] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, volume 23, 2010.