Efficiency and Generalization of Sparse Neural Networks

by

Elena-Alexandra Peste

May, 2023

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Maria Ibáñez, Chair
Christoph H. Lampert
Dan Alistarh
Marco Mondelli
Dmitry Vetrov

The thesis of Elena-Alexandra Peste, titled *Efficiency and Generalization of Sparse Neural Networks*, is approved by:

Supervisor: Christoph H. Lampert, ISTA, Klosterneuburg, Austria

Signature: _____

Co-supervisor: Dan Alistarh, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Marco Mondelli, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Dmitry Vetrov, HSE University, Moscow, Russia

Signature: _____

Defense Chair: Maria Ibáñez, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Elena-Alexandra Pește
May, 2023

Signed page is on file

Abstract

Deep learning has become an integral part of a large number of important applications, and many of the recent breakthroughs have been enabled by the ability to train very large models, capable to capture complex patterns and relationships from the data. At the same time, the massive sizes of modern deep learning models have made their deployment to smaller devices more challenging; this is particularly important, as in many applications the users rely on accurate deep learning predictions, but they only have access to devices with limited memory and compute power. One solution to this problem is to *prune* neural networks, by setting as many of their parameters as possible to zero, to obtain accurate sparse models with lower memory footprint. Despite the great research progress in obtaining sparse models that preserve accuracy, while satisfying memory and computational constraints, there are still many challenges associated with efficiently training sparse models, as well as understanding their generalization properties.

The focus of this thesis is to investigate how the training process of sparse models can be made more efficient, and to understand the differences between sparse and dense models in terms of how well they can generalize to changes in the data distribution. We first study a method for co-training sparse and dense models, at a lower cost compared to regular training. With our method we can obtain very accurate sparse networks, and dense models that can recover the baseline accuracy. Furthermore, we are able to more easily analyze the differences, at prediction level, between the sparse-dense model pairs. Next, we investigate the generalization properties of sparse neural networks in more detail, by studying how well different sparse models trained on a larger task can adapt to smaller, more specialized tasks, in a transfer learning scenario. Our analysis across multiple pruning methods and sparsity levels reveals that sparse models provide features that can transfer similarly to or better than the dense baseline. However, the choice of the pruning method plays an important role, and can influence the results when the features are fixed (linear finetuning), or when they are allowed to adapt to the new task (full finetuning). Using sparse models with fixed masks for finetuning on new tasks has an important practical advantage, as it enables training neural networks on smaller devices. However, one drawback of current pruning methods is that the entire training cycle has to be repeated to obtain the initial sparse model, for every sparsity target; in consequence, the entire training process is costly and also multiple models need to be stored. In the last part of the thesis we propose a method that can train accurate dense models that are compressible in a single step, to multiple sparsity levels, without additional finetuning. Our method results in sparse models that can be competitive with existing pruning methods, and which can also successfully generalize to new tasks.

Acknowledgements

I would like to begin by thanking my supervisors, Dan and Christoph, for teaching me so much about research, while also giving me the freedom to find and pursue my research interests. I am grateful for their support and for believing in me throughout this journey. I also thank Marco and Dmitry for being part of my thesis committee, and for their valuable feedback on my PhD projects, and I thank Maria for being my defence chair.

Being part of two research groups meant I was also lucky to have more colleagues to collaborate and exchange ideas with. I would like to thank all the current and former members of the Alistarh and Lampert groups, and to all the interns and visitors I had the opportunity to meet during my PhD. I have learned a lot from all of you and I wish you all the best in your research careers! Special thanks go to my collaborators Jen, Adrian, and Eldar, for their dedication and for the fun projects we shared together.

I am also grateful for the friends I met at ISTA, and for all the nice memories we shared, which made the PhD journey feel less lonely. Thank you Michelle, Linda, Djordje, Sarath, Fleur and Alex for all the fun times, and I hope we will continue making good memories!

In the years spent working towards my PhD, I have learned that the biggest challenge was in learning to accept myself, with all my imperfections. For this, I am deeply grateful for having close people who cared for me unconditionally, independent of any results or achievements. I thank my parents, my brother and my grandparents, for their love, for always believing in me and supporting my dreams. I also thank my best friends, Alexandra and Miruna, for their continuous support and understanding; even if we are far from each other, I know you are only a phone call away. Lastly, I am grateful to the most important person, Catalin, for his unconditional love and for teaching me how to accept and love myself. This thesis is dedicated to him, and I can't wait for the rest of our journey together!

Funding sources. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385, and from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 805223 ScaleML). Furthermore, the candidate acknowledges the support from the Scientific Service Units (SSU) of ISTA through resources provided by Scientific Computing (SciComp).

About the Author

Alexandra Peste obtained her BSc and MSc in mathematics from the University of Bucharest. She joined ISTA in September 2018, where she was supervised by Dan Alistarh and Christoph Lampert. Her main research interests are related to sparsity in deep learning. During her PhD, she has also explored topics related to data unlearning and data compression for machine learning models. Her work, which was published in several international conferences, focused on developing efficient methods for obtaining accurate sparse neural networks, with a focus on their generalization properties.

List of Collaborators and Publications

This thesis is based on the following first-author publications of Elena-Alexandra Pește. We provide a summary of the contributions of each author, referred to by their initials.

1. Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating Compressed/DeCompressed Training of Deep Neural Networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021

   – Chapter 3 is based on this paper.
   – All authors participated in the discussions which lead to the development of the main algorithm.
   – AP and EI worked on the implementation of the algorithm.
   – AV developed and wrote the theoretical analysis.
   – Throughout the project, DA supervised AP and EI, coordinated regular progress meetings and suggested related work.
   – AP performed and analyzed the main experiments on large scale image classification, language modelling and memorization.
   – EI performed additional experiments, including the model output comparison.
   – All authors contributed to the writing of the final manuscript.

2. Eugenia Iofinova, Alexandra Peste, Mark Kurtz, and Dan Alistarh. How well do sparse ImageNet models transfer? In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022

   – Chapter 4 is based on this paper.
   – AP and EI contributed equally to this project (joint first authorship).
   – DA initially proposed the problem to AP and EI, suggested related work, supervised AP and EI throughout the entire project, and proposed improvements and additional experiments.
   – EI provided the initial implementation of the framework, as well as preliminary results, and was involved in later improvements and extensions to the code base.
   – AP extended the initial implementation, proposed and developed improvements to the experimental framework, and explored related directions relevant to the project.
   – AP and EI jointly performed all the linear and full finetuning experiments.
   – MK provided an integration with the DeepSparse [KKG+20, Dee21] framework to obtain practical speed-ups, and performed the object detection and segmentation experiments.

- AP worked on the initial analysis of the results, performed auxiliary experiments, and proposed improvements to the analysis.
- EI developed the main framework for the analysis of the results, and produced the main graphs and tables.
- AP and EI wrote most of the paper, DA wrote the introduction and was extensively involved in the editing of the paper.

3. Alexandra Peste, Adrian Vladu, Eldar Kurtic, Christoph H. Lampert, and Dan Alistarh. CrAM: A Compression-Aware Minimizer. *International Conference on Learning Representations (ICLR)*, 2023

   - Chapter 5 is based on this paper.
   - AP and DA jointly formulated the problem and proposed the research objectives.
   - AV provided feedback and improvements on the initial formulation, framed the problem in the context of robust optimization, developed and wrote the theoretical analysis.
   - AP implemented and improved the method, performed and analyzed the image classification and sparse transfer experiments, as well as the ablation study.
   - CHL discussed with AP the initial version of the algorithm, suggested improvements, and provided feedback on the results, as well as on the manuscript.
   - DA supervised AP throughout the project, suggested new experiments, particularly on language modelling, which were discussed with EK.
   - EK performed the language modelling experiments and provided the analysis of the results in the paper.
   - AP wrote the initial version of the paper, and edited and improved the manuscript, together with DA.

The candidate has also co-authored the following publications and manuscripts, which are not included in the thesis:

- Eugenia Iofinova, Alexandra Peste, and Dan Alistarh. Bias in Pruned Vision Models: In-Depth Analysis and Countermeasures. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023
- Alexandra Peste, Dan Alistarh, and Christoph H Lampert. SSSE: Efficiently Erasing Samples from Trained Machine Learning Models. *arXiv preprint arXiv:2107.03860*, 2021 (a shorter version of the manuscript also appeared in the NeurIPS 2021 Workshop Privacy in Machine Learning)
- Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research (JMLR)*, 2021

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

In recent years, we have witnessed the tremendous progress made by deep neural networks in solving diverse tasks, spanning multiple fields, such as computer vision [HZRS16, DBK+21], natural language processing [VSP+17, DCLT19], or reinforcement learning [MKS+15, SSS+17]. Neural networks are now capable of generating complex images from text descriptions [RDN+22], entertaining realistic conversations with humans [Ope22], discovering new protein structures [TAW+21] and even outplaying world champions at complex games [SSS+17]. Along with the increased research efforts, these incredible achievements have been enabled by the availability of specialized hardware with improved memory and computational support.

While undeniably impressive, these highly performing neural networks have an important drawback: they require massive memory and compute power for both training and inference, as they often have many millions or even billions of parameters. For example, the GPT-3 model [BMR+20], which is a popular pre-trained generative model from the Transformer [VSP+17] family, used for natural language processing tasks, has 175 billion parameters and requires more than 300GB of memory for storage, which exceeds the capacity of any currently available graphics processing unit (GPU) [FAHA23]. The sheer size of these models, and the resources they require pose important questions regarding both their environmental and financial costs, and several researchers have raised these concerns [SGM20, BGMMS21]. Furthermore, this trend of designing highly performing deep learning architectures with massive sizes is at odds with the efforts of making the benefits deep learning more accessible, particularly on hardware with fewer computational resources. Ideally, neural networks should be deployable on edge devices, such as mobile phones or personal laptops, to enable more users to benefit from the advances in deep learning.

To address these challenges, a great amount of research interest has been dedicated in recent years to neural network compression, and two popular approaches have been in the center of these efforts: pruning and quantization. The quantization techniques [GKD+21] concentrate on reducing the precision of the model, by restricting the set of possible values of the parameters. On the other hand, pruning methods [HABN+21] focus on setting as many model parameters as possible to zero, while still ensuring good accuracy; the resulting sparse models would in turn have reduced memory requirements, as their parameters can be stored in sparse format.

While both approaches have been proven very successful at reducing the size of deep models, and in fact they can be used complementarily, this thesis focuses on compression methods based on *pruning techniques*. Our focus on model pruning is motivated by the great interest in this field,

which has witnessed the development of a myriad of techniques for obtaining sparse models, many of them presented in the survey of [HABN+21]. Moreover, there is increasingly available hardware support for sparse models [Dee21, Gra21], which makes them of great practical interest. From a theoretical point of view, sparse models can reveal or help us understand some interesting generalization properties of deep networks [AGNZ18, FC19, JCR+22]. Still, there are many things to improve when it comes to reducing the training costs of sparse models, together with narrowing the performance gap between sparse and dense models, and understanding the differences between them. In particular, in this thesis we study ways of reducing the computational costs associated with training sparse models, together with better understanding their generalization properties, by comparing how well they can be used to adapt to new data distributions or how much they differ from dense models, in terms of predictions.

One important challenge regarding pruning methods concerns the training costs for obtaining accurate sparse models. Despite the fact that pruning methods can substantially reduce the memory footprint of deep learning models, some of the most popular pruning methods [ZG17, SA20, FKA21, KCN+22] still require important memory and computational resources for training; for example, they often start from a pretrained dense model, and only gradually increase the sparsity level, until a desired target level is reached. With these considerations in mind, we study methods meant to tackle the challenge of reducing the training costs of sparse models. Namely, we develop a method for training sparse models at a lower training cost compared to that of a dense model. Our method has the additional advantage of allowing the co-training of accurate sparse–dense model couples, with a single training cycle.

Another challenge in reducing the costs of obtaining accurate sparse models is that often the training process has to be repeated for each sparsity target; this in turn can be quite cumbersome, as multiple hyper-parameter search rounds might be needed; additionally, this does not allow for enough flexibility, particularly in applications where users might want to switch between multiple sparse models, depending on their computational needs. For this reason, we also investigate in this thesis methods for training accurate dense models that are *prunable* in a single step, to multiple target sparsity levels, without additional finetuning.

An equally important research problem for pruning methods concerns their generalization abilities. While several works have discussed the potential of sparse models for generalization [AGNZ18, JCR+22], there is still more to be understood regarding the ability of sparse models to adapt to different data distributions, as well as the practical differences between sparse and dense models. In this thesis, we approach these challenges by examining how well existing pruning methods can be used for transfer learning, that is, to leverage information learned on a larger, more general dataset, to smaller, more specialized tasks. Additionally, we show that the methods we developed for training prunable models have the flexibility to generalize to multiple tasks, when models are pruned to different sparsity levels.

Given the above mentioned challenges in the field of neural network pruning, we organize the thesis as follows:

- In Chapter 2 we provide an overview of the most popular existing pruning methods, and we also present the main architectures and datasets used for the methods proposed.
- In Chapter 3 we present a method for co-training accurate sparse and dense models, which alternates dense training steps with optimization in the sparse support. Thus, we are able to obtain sparse models at a lower computational cost compared to training dense models. Additionally, we show theoretical guarantees for the convergence of our

method and also analyze its generalization properties, by examining differences between co-trained sparse and dense models, at sample prediction level. This chapter is based on our published work [PIVA21].

- In Chapter 4 we investigate a practical use-case of sparse models; namely, how amenable different pruning methods are to finetuning on smaller tasks. This chapter, based on the published work [IPKA22], concerns both the computational and generalization aspects of neural network pruning. On one hand, sparse models can be used for finetuning on edge devices, due to their reduced memory and computation requirements, and on the other hand, we investigate the generalization capabilities of sparse models to changes in the data distribution.

- In Chapter 5, which is based on [PVK+23], we again tackle, from a different angle, the computational and generalization aspects concerning sparse models. Namely, we introduce a method for training dense models that can be easily pruned in a single step, to different sparsity levels, without additional finetuning; this would in turn remove the need for training a separate model from scratch for every desired sparsity level, and would ultimately reduce the overall cost of training multiple sparse models. Additionally, our approach is inspired by methods that enforce flatter loss surfaces [FKMN21], which are believed to lead to better generalization, through flatter local minima [HS97a, KMN+17, LXT+18].

- Lastly, in Chapter 6 we look back at our methods and results, and consider new research directions to explore in the future.

CHAPTER 2

Background

In this chapter we present basic concepts regarding training and pruning neural networks, which will serve as background for the subsequent chapters. In the first part, we discuss the most important concepts regarding training neural networks, including optimization, different types of architectures, as well as common datasets used for benchmarking. We further discuss additional topics of interest, such as transfer learning, and in particular linear and full finetuning. The second part of the chapter is dedicated to neural network pruning, and will describe some of the most widely used methods for obtaining sparse models, together with some of the challenges associated with them.

2.1 Fundamentals of Neural Network Training

In this section we describe the fundamental concepts regarding machine learning, focused on the training of neural networks.

2.1.1 Notations and Overview

We describe some basic concepts in machine learning, which also apply to deep learning. In a nutshell, machine learning represents the collection of methods designed for finding patterns in given finite data, through an automatic process, with the purpose of inferring predictions on new, unseen data. A machine learning model can be described as a mapping which takes the input data and returns the corresponding predictions. Neural networks, in particular, represent a large class of machine learning models and the field concerned with the study of neural networks is known as *deep learning*.

The process of optimizing the machine learning model on the given data–referred to as *training data*– is suggestively called *learning* or *training*. The goal of the training process is to provide meaningful predictions on new *test* data. There are different types of learning paradigms in machine learning, and a common categorization takes into account the presence or absence of labels for the training data. Since most of our methods and results are presented for supervised problems for which the ground-truth labels are available, we describe the basic framework for supervised learning.

Assume our data and labels reside in the sets $\mathcal{X} \subseteq \mathbb{R}^d$, and $\mathcal{Y}$, respectively, on which we further assume a probability distribution $\mathcal{D} \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$. In the case of classification problems, $\mathcal{Y}$ is

discrete, e.g. for binary classification $\mathcal{Y} = \{0, 1\}$. The training set D consists of i.i.d. pairs $\{(x_1, y_1), \dots (x_n, y_n)\}$ sampled from $\mathcal{D}$. A hypothesis class $\mathcal{H}$ represents a set of functions, or *models*, with parameters in $\Theta \subseteq \mathbb{R}^N$, which map inputs from $\mathcal{X}$ to the predictions in $\mathcal{Y}$. Namely, $\mathcal{H} = \{h_{\boldsymbol{\theta}} : \mathcal{X} \longrightarrow \mathcal{Y} | \boldsymbol{\theta} \in \Theta\}$. For example, for linear models and binary classification, $h_{\boldsymbol{\theta}}(x) = \mathbb{1}[\boldsymbol{\theta}^T x \geq 0]$. The goal is to find the hypothesis $h \in \mathcal{H}$ which achieves the best representation of the training data. In the case of supervised learning, we formalize this through the empirical risk defined as $L_D(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i)$, where $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow [0, \infty)$ is the loss function measuring how close our model's predictions are from the real labels, for example the 0/1 loss $\ell(y', y) = \mathbb{1}[y' \neq y]$. We typically perform *empirical risk minimization* to find the best hypothesis $h^* = \arg\min_{h \in \mathcal{H}} L_D(h)$.

The goal is to obtain a model that *generalizes* well, i.e. gives good predictions on new data. This can be expressed through the expected risk $L_{\mathcal{D}}(\boldsymbol{\theta}) = \mathbb{E}_{x,y \sim \mathcal{D}}[\ell(h_{\boldsymbol{\theta}}(x), y)]$. However, in practice we do not usually have access to the true data distribution, but instead have a *test set* $D' = \{(x'_i, y'_i), \dots (x'_m, y'_m)\}$ assumed to be drawn from the same underlying distribution $\mathcal{D}$, and we measure the quality of our empirical risk minimizer on D'.

While in the field of learning theory [SSBD14], the empirical risk is typically defined for binary classification problems, in which the loss function is the 0/1 loss, we expand this definition to different types of problems, and also to allow differentiable loss functions. For example, for multiclass classification problems, where $\mathcal{Y} = \{1, 2, \dots K\}$, it is typical to have a one-hot encoded representation of the targets into a vector in $\{0, 1\}^K$ where all components are zero, except for the class being represented. Then, the model h outputs for each sample x a *prediction vector* $\tilde{y}$, which consists of the probabilities for each class. A very common loss function is the cross-entropy $\ell : C^K \times C^K \longrightarrow [0, \infty)$, where $C^K \subseteq [0, 1]^K$ is the probability simplex, and $\ell(\tilde{y}, y) = -\sum_{k=1}^{K} y^{(k)} \log \tilde{y}^{(k)}$. Since most often the models are parameterized by some $\boldsymbol{\theta} \in \Theta$, we use throughout the thesis $f_i(\boldsymbol{\theta})$ or $\ell_i(\boldsymbol{\theta})$ to refer to $\ell(h_{\boldsymbol{\theta}}(x_i), y_i)$, where ℓ is typically the cross-entropy function. Then, the training loss is defined as

$$L_D(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_{\boldsymbol{\theta}}(x_i), y_i) \qquad (*)$$

Equivalently, we will also use the notation $f(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\theta})$ to refer to the training loss function, particularly when we focus on optimization aspects. While our goal is to find $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \Theta} L_D(\boldsymbol{\theta})$, there are two possible phenomena that could occur: *underfitting*, which is insufficient learning of the training set, or *overfitting*, when the model learns the training set very well, but performs very badly on the test set. Underfitting can be mitigated by increasing the complexity of the model, while to avoid overfitting it is common to add different regularizations to the model, for example ℓ_2 regularization of $\boldsymbol{\theta}$, also known as weight decay in the case of neural networks, which can be expressed through the regularized loss function $L_{D,\lambda}(\boldsymbol{\theta}) = L_D(\boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2$. Moreover, to avoid overfitting, a good practice is to use a separate *validation set* (typically a random subset of the original training set) during model selection, and to only use the test set for the final evaluation.

In the next section, we will explore different algorithms for finding $\boldsymbol{\theta}^* = \arg\min L_D(\boldsymbol{\theta})$, by exploring different optimization algorithms for differentiable loss functions.

2.1.2 Optimization

In this section we describe some of the basic optimization algorithms that are used for training machine learning models, and neural networks in particular. In what follows, for notation

simplicity we denote the loss function $f : \mathbb{R}^N \longrightarrow \mathbb{R}$, which is assumed to be differentiable. We begin by introducing some commonly used definitions, which will also serve as simplifying assumptions.

Definition 2.1.1. *A differentiable function $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ is defined as*

- *ℓ-strongly convex if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$, $f(\boldsymbol{x}) - f(\boldsymbol{y}) \leq \nabla f(\boldsymbol{x})^T(\boldsymbol{x} - \boldsymbol{y}) - \frac{\ell}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2$*
- *L-smooth if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$, $f(\boldsymbol{x}) - f(\boldsymbol{y}) \leq \nabla f(\boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y}) + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2$*

It is well-known that L-smoothness is equivalent to having L-Lipschitz gradients, i.e. $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$.

Gradient Descent

One of the most basic gradient-based optimization algorithms for minimizing function f is *gradient descent* (GD), which finds a minimizer $\boldsymbol{\theta}_T$, by performing the iteration

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t),$$

for $t = 0, 1, \ldots T$ and initial point $\boldsymbol{\theta}_0$.

It is well-known [Bub15] that for ℓ-strongly convex and L-smooth functions, GD converges to a point $\boldsymbol{\theta}_T$ close to the optimum $\boldsymbol{\theta}^*$, such that

$$\|\boldsymbol{\theta}_T - \boldsymbol{\theta}^*\|^2 \leq \exp\left(-\frac{T}{\kappa}\right)\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|^2,$$

where $\eta = \frac{1}{L}$ and $\kappa = \frac{\ell}{L}$. In other words, GD converges to a point that is ϵ-close to the optimum in $O(\ln \frac{1}{\epsilon})$ iterations.

While GD has good convergence rates for strongly convex and smooth functions, this also comes with quite a high computational cost. For example, as previously discussed, in machine learning the loss function typically has the form $f(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{\theta})$; then, the cost of a GD iteration is $O(nG)$, where G is the cost of computing a gradient $\nabla f_i(\boldsymbol{\theta})$. In the case of neural networks, both the size of the dataset and the size of the model are very large, which makes the computation of the full gradient $\nabla f(\boldsymbol{\theta})$ at each iteration extremely costly.

Stochastic Gradient Descent

One solution to mitigate the costs of a GD iteration is to use instead an unbiased estimator of the gradient. This can be achieved in practice by sampling uniformly at random, at each iteration, a component $f_i(\boldsymbol{\theta})$ and using its gradient instead to update $\boldsymbol{\theta}$. Thus, we define the *stochastic gradient descent* (SGD) at iteration $t + 1$ as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t g(\boldsymbol{\theta}_t), \tag{SGD}$$

where $g(\boldsymbol{\theta}_t)$ is an unbiased estimator of the full gradient $\nabla f(\boldsymbol{\theta})$ (i.e. $\mathbb{E}[g(\boldsymbol{\theta}_t)] = \nabla f(\boldsymbol{\theta}_t)$).

It is common to assume that the stochastic gradient estimation has bounded variance, i.e. there exists $\sigma > 0$ such that $\mathbb{E}[\|g(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta})\|^2] \leq \sigma^2$, for any $\boldsymbol{\theta} \in \mathbb{R}^N$. Then, under the same assumptions as before (ℓ-strong convexity and L-smoothness), for $\eta_t = \frac{1}{2L}$, it can be shown (as for example in [GLQ+19]) that SGD converges after $O(\ln \frac{\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|^2}{\epsilon})$ iterations to

a point $\boldsymbol{\theta}_T$ such that $\mathbb{E}[\|\boldsymbol{\theta}_T - \boldsymbol{\theta}^*\|^2] \leq \epsilon + \frac{\sigma^2}{\ell L}$. More generally, when f is only convex and L-smooth, one can obtain a bound on the function evaluated at the average of the SGD updates, as presented in Theorem 6.3 from [Bub15]. While the convergence of SGD is slower, in terms of number of iterations, compared to GD, its cost per iteration makes it a good choice for problems for which the dataset and the model are large.

In practice, one variant of SGD that is very widely used is *mini-batch SGD*. Instead of sampling one component f_i at each iteration, we sample a subset $f_{i_1}(\boldsymbol{\theta}), \dots f_{i_B}(\boldsymbol{\theta})$ and use the average gradient $g(\boldsymbol{\theta}) = \frac{1}{B}\sum_{j=1}^{B} \nabla f_{i_j}(\boldsymbol{\theta})$ to compute the next iteration. Then, the variance of the stochastic gradient is bounded by $\frac{\sigma^2}{B}$, from which it follows that mini-batch SGD requires $\frac{1}{B}$ times fewer iterations for convergence; this can be more easily seen by examining Theorem 6.3 from [Bub15]. The mini-batch SGD is the most popular algorithm for optimizing neural networks, since it still has a low cost per iteration, as the computation of the gradients in the mini-batch can be parallelized on GPUs.

Momentum. Additional modifications of gradient based GD or SGD algorithms have been proposed to improve their convergence rates. One popular improvement is the use of momentum. The main motivation is that SGD might converge very slowly or its iterations might oscillate a lot, when the loss function is steeper along some directions, compared to others [Sut86]. For this reason, it is common to use *momentum* [Pol64, RHW86a, Qia99, Rud16], for which an iteration is defined as follows:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta g(\boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + v_{t+1}, \end{aligned} \qquad \text{(SGD with momentum)}$$

where $g(\boldsymbol{\theta}_t)$ is an unbiased gradient estimator, μ is the momentum value, η is the learning rate, v_t is the momentum vector, with $v_0 = 0$. Other types of momentum can be defined, such as Nesterov momentum [Nes83, BBLP13, SMDH13], which also improve the convergence rates of GD or SGD. In general, for the experiments in this thesis, we mostly used SGD with momentum.

Additional algorithms. While SGD (with momentum) is one of the most popular algorithms used for optimizing machine learning models, and neural networks in particular, multiple other algorithms have been introduced, such as for example Adagrad [DHS11] or Adam [KB15], which compute individual learning rates for each parameter in the model. We briefly describe *Adam*, which we also used in some of our experiments on language models. Adam computes individual learning rates for each parameter, by taking into account the first and second order moments of the gradients. Namely, for hyperparameters β_1 and β_2, called exponential decay rates, and for $g(\boldsymbol{\theta}_t)$ unbiased gradient estimator, the Adam update is computed as follows (where $\odot$ is the Hadamard product performing element-wise multiplication):

$$\begin{aligned} m_{t+1} &= \beta_1 \cdot m_t + (1-\beta_1) \cdot g(\boldsymbol{\theta}_t) \\ v_{t+1} &= \beta_2 \cdot v_t + (1-\beta_2) \cdot g(\boldsymbol{\theta}_t) \odot g(\boldsymbol{\theta}_t) \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^{t+1}} \quad \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \cdot \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1} + \epsilon}} \end{aligned} \qquad \text{(Adam update)}$$

2.1.3 Deep Learning Architectures

While artificial neural networks have been introduced a long time ago [MP43], they have only become very popular in recent years [KSH17], through their remarkable ability to solve diverse tasks, greatly surpassing previous models. Their popularity can also be observed through the great research interest in the field, with many different new architectures proposed regularly. One common characteristic of these models, however, is the presence of a typically large parameter space, disposed in multiple layers, which ultimately creates a very complex intractable model. We describe in this section some of the most popular types of deep learning architectures.

Fully Connected Neural Networks

In general, a feed-forward neural network $h_{\boldsymbol{\theta}} : \mathcal{X} \subseteq \mathbb{R}^d \longrightarrow \mathcal{Y}$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^N$ comprises of a composition of multiple simpler functions $h^{(\ell)}_{\boldsymbol{\theta}^\ell} : \mathbb{R}^{N_{\ell-1}} \longrightarrow \mathbb{R}^{N_\ell}$, with $\ell \in \{1, \dots L\}$, each parameterized by $\boldsymbol{\theta}^\ell$, such that $\boldsymbol{\theta} = (\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^L)$. Each of the components $h^{(\ell)}$ constitutes a layer with parameters $\boldsymbol{\theta}^\ell$.

Thus, for any input $\boldsymbol{x} \in \mathcal{X}$,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = h^{(L)}_{\boldsymbol{\theta}^L}(h^{(L-1)}_{\boldsymbol{\theta}^{L-1}}(\dots h^{(1)}_{\boldsymbol{\theta}^1}(\boldsymbol{x})))$$

Typically, each $h^{(\ell)}_{\boldsymbol{\theta}^\ell}$ is itself comprised of a linear function and a non-linearity: $h^{(\ell)}_{\boldsymbol{\theta}^\ell}(z) = \sigma(\tau_{\boldsymbol{\theta}^\ell}(z))$, where $\tau_{\boldsymbol{\theta}^\ell} : \mathbb{R}^{N_{\ell-1}} \longrightarrow \mathbb{R}^{N_\ell}$ is a linear function: $\tau_{\boldsymbol{\theta}^\ell}(z) = w^\ell \cdot z + b^\ell$ (with $\boldsymbol{\theta} = (w, b)$) and $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ is a non-linear function applied component-wise to each element of $\tau_{\boldsymbol{\theta}^\ell}(z)$. For the particular case when each $\tau_{\boldsymbol{\theta}^\ell}$ is a linear function, this type of architecture is called *fully connected neural network* (FCNN). We note that each individual component $(h_{\boldsymbol{\theta}^\ell})_i$ in a layer is called a *neuron*, and is connected to all the neurons from the previous layer $h^{(\ell-1)}$ via $\boldsymbol{\theta}^\ell$, therefore the name fully connected. Another common name for fully connected neural networks is multi layer perceptrons (MLPs).

Non-linearities. As previously mentioned, it is necessary to combine linear functions with non-linearities in the definition of the neural network, to encourage more complexity in the representation of the data, since otherwise the model would collapse into a linear one. One of the most widely used non-linearities is the Rectified Linear Unit (ReLU) [JKRL09, NH10], defined as $\sigma(x) = \max(0, x)$, which is present in many of the modern deep learning architectures [HZRS15]. Other popular functions include the sigmoid or hyperbolic tangent. The non-linear functions are commonly known as *activation functions*. In the case of multi-class classification problems, it is common to use the softmax function $\sigma(\boldsymbol{x}) = (\frac{\exp \boldsymbol{x}_i}{\sum_j \exp \boldsymbol{x}_j})_i$ as the final non-linearity, to output a probability distribution over the possible classes, and to train the model using the cross-entropy loss defined in the previous section.

Backpropagation. The most common way to train neural networks is to optimize their parameters on the train data using a stochastic gradient-based algorithm (e.g. SGD with momentum, or other algorithm described in the previous section). For this, we need to compute the gradient (or an unbiased gradient estimator) of the training loss defined in (*). The algorithm used for computing the gradients of a neural network is *backpropagation* [RHW86a]; it is based on the chain rule, and it sequentially computes the gradients of the loss with respect to the parameters in each layer, starting from the final one, and passing the Jacobian with respect to the input of each layer to its predecessor.

Universal function approximators. Despite their apparent simplicity, fully connected neural networks have been shown to be universal function approximations: in particular, it is well known [Cyb89, HSW89, Fun89, Hor91] that neural networks with one hidden layer, arbitrary width and suitable activation function can approximate any continuous function on a compact domain; more recently, a similar result has been shown for fully connected neural networks with bounded width [LPW+17].

Convolutional Neural Networks

While fully connected neural networks are universal function approximators, they have some drawbacks: they are known to be difficult to train for more complex problems, particularly since they can have a very large number of parameters. One example where fully connected neural networks do not work well is for image classification. Images are two-dimensional objects, and flattening them to a one-dimensional vector in order to apply a linear transformation would break their structure, for example the similarity of neighboring pixels. For this reason, *convolutional neural networks* (CNNs) have been introduced [FMI83, LDS90], which replace the fully connected layers in a neural network with *convolutional layers*.

We define a two-dimensional convolution operation as follows. Assume a two-dimensional input $x \in \mathbb{R}^{w \times h}$, parameters or *kernel* $\boldsymbol{\theta}^{\mathbb{R}^{k_w \times k_h}}$ and *strides* $(s_w, s_h) \in \{1, \dots m\} \times \{1, \dots n\}$. Then, the convolution $\tau_{\boldsymbol{\theta}}$ maps the input $\boldsymbol{x} \in \mathbb{R}^{w \times h}$ to an output $\tilde{\boldsymbol{x}} \in \mathbb{R}^{(\lfloor \frac{w-k_w}{s_w} \rfloor + 1) \times (\lfloor \frac{h-k_h}{s_H} \rfloor + 1)}$ by the following rule, as used in neural networks:

$$\tau_{\boldsymbol{\theta}}(\boldsymbol{x}) = \left(\sum_{p=1}^{k_w} \sum_{q=1}^{k_h} \boldsymbol{\theta}_{p,q} \cdot \boldsymbol{x}_{(i-1)\cdot s_w + p, (j-1)\cdot s_h + q} \right)_{i,j}$$

Usually, the input to a convolution layer is three dimensional; thus, the convolution $\tau_{\boldsymbol{\theta}}$ typically takes an input $\boldsymbol{x} \in \mathbb{R}^{I \times W \times H}$ and outputs a tensor in $\mathbb{R}^{O \times (\lfloor \frac{w-k_w}{s_w} \rfloor + 1) \times (\lfloor \frac{h-k_h}{s_h} \rfloor + 1)}$, where I and O are the number of input and output channels, respectively. Then, the parameters are $\boldsymbol{\theta} \in \mathbb{R}^{I \times k_W \times k_H \times O}$, and the kernels are three dimensional sub-tensors $\boldsymbol{\theta}_{\cdot,\cdot,\cdot,o}$. The operation is very similar to the one described above, except that in this case the results of each individual two-dimensional convolution are summed over the input channels, and stacked over the output channels. It is easy to see that, compared to fully connected layers, the kernel of a convolutional layer is shared across the features in a channel; this in turn greatly reduces the number of parameters of a CNN, compared to FCNN on the same input data.

Pooling layers. In the architecture of a CNN it is common to find other types of layers, used to aggregate the features present in a region, and to reduce the dimension of the feature maps, i.e. of the width and height of the output $\tilde{\boldsymbol{x}}$. These operations form *pooling layers*, which can be defined very similarly to the convolutional one, except that instead of performing the dot product with a kernel, we compute the average or maximum over a region. This results in two very popular choices of the pooling layers, namely average and max pooling, respectively.

Residual connections. One common problem of CNNs in particular, but also of other types of architectures, is that it is generally hard to train *very deep* models, i.e. with many layers. Many of the challenges arise from the behaviour of the gradients, which can be very small and lose information from earlier layers –*vanishing gradients*–, or they can become very large, making the training process unstable –*exploding gradients*. One method that can greatly help with these problems, by making the training process seemingly smoother, is the use of

residual layers or *skip connections* [HZRS15, HZRS16]. In a nutshell, a residual connection takes an input $\boldsymbol{x}$ to a subsequent layer F, and returns $F(\boldsymbol{x}) + \boldsymbol{x}$. It is common to connect via skip connections layers that are further apart, or to use linear functions instead of the identity map, to ensure the dimensions match. Basically, skip connections create direct paths between later and earlier layers, which allow the gradients to flow easier. Residual connections have revolutionized the training of CNNs, and *Residual Networks* (ResNets) [HZRS15] have become very popular among practitioners. We also use them in many of the experiments presented in this thesis, in Chapters 3, 4 and 5. Moreover, the benefits of residual connections are also present in other architectures, for example Transformer models [VSP+17] used in natural language processing.

Normalization layers. Another important component in the success recipe of training a neural network consists of the use of normalization layers. One of the difficulties associated with training neural networks is the fact that the input distribution to each layer changes as parameters are optimized, fact known as *internal covariate shift* [IS15]. This phenomenon can negatively impact the convergence of the model, as inputs to a layer can reside inside regions where the activation function (e.g. sigmoid) saturates, which would lead to vanishing gradients. Inspired by the well-known benefits of normalizing the input data to a model [LBOM12], one very successful method proposed to help mitigate the above mentioned issues is *Batch Normalization* (BatchNorm or BN) [IS15]. In a nutshell, BatchNorm works by simply normalizing the inputs to a layer, followed by applying a linear transformation. Assume $x^{(\ell)}$ is the input to a layer ℓ, computed for a mini-batch B of data samples; the BatchNorm layer computes the following transformation, before applying non-linearities and passing the result to the next layer: $\hat{x}^{(\ell)} = \lambda^{(\ell)} \cdot \frac{x^{(\ell)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta^{(\ell)}$, where $\mu_B = \frac{1}{B}\sum_{i=1}^{B} x_i^{(\ell)}$ and $\sigma_B = \frac{1}{B}\sum_{i=1}^{B}(x_i^{(\ell)} - \mu_B)^2$ are the mini-batch statistics. The per-layer scaling parameters $\lambda^{(\ell)}$ and $\gamma^{(\ell)}$ are optimized alongside the weights of the model, and are used to allow more flexibility in the representation learned by each layer. To estimate the statistics over the entire data distribution, it is standard to keep a running mean over the means and standard deviations for each mini-batch observed during training; then, at inference time, we normalize each layer using these statistics $\hat{\mu}$ and $\hat{\sigma}$. It has been initially shown in [IS15] that BatchNorm layers can substantially improve the training of neural networks, both in terms of convergence time and final accuracy; since then, BatchNorm layers have been widely adopted in most of the modern architectures, particularly those used for computer vision applications. Furthermore, it is well-accepted [IS15] that the use of BatchNorm decreases the need to use other regularization layers, such as Dropout [SHK+14].

Popular Computer Vision Datasets. We now describe some of the most popular datasets used for benchmarking the performance of CNNs, which we also use for many of the experiments presented in this thesis. One of the most common datasets is CIFAR-10 [KH+09], which consists of 50,000 train samples and 10,000 test samples. It contains colored images of 32×32 pixels, from 10 different classes representing various objects or animals, with equal number of samples between classes. It is usually one of the easiest datasets used for testing the performance of various algorithms on image classification problems, and it is typically trained using a variant of the ResNet architecture [HZRS15]. A more complex dataset is CIFAR-100 [KH+09], which has a similar format in terms of size, but contains instead images from 100 classes. Likewise, it is trained using a ResNet or a WideResNet [ZK16] architecture.

One of the first large-scale computer vision datasets introduced was ImageNet [DDS+09], which in its full version contains more than 14 million colored images from more than 20,000 classes.

A smaller version of this dataset was released [RDS+15], alongside the public competition ImageNet Large Scale Visual Recognition Challenge (ILSVRC); this trimmed-down version of ImageNet consists of 1.28 million training images belonging to 1000 classes, together with additional 50,000 validation samples, and 100,000 test samples that are used for the final evaluation during the competition. The challenge gained a lot of popularity when it featured the first CNN [KSH12, KSH17], later known as AlexNet, to surpass existing methods, as well as human performance. Nowadays, ImageNet, in its ILSVRC variant, is still a challenging dataset and is widely used for benchmarking the performance of neural network architectures or training methods. In this thesis, we use this version of ImageNet to evaluate the performance of our algorithms, and present the Top-1 accuracy on the 50,000 samples validation set, as is the common practice in the field.

Transformer Models

Natural Language Processing (NLP) tasks have long been a challenge for deep learning models, due to the inherent temporal dependencies present in the data. Some of the most successful early attempts at language modelling have built the temporal dependencies inside the neural network through recurrent layers, for which the hidden state of an input from a sequence depends on the hidden states of all the previous elements from the sequence; these models have been traditionally known as recurrent neural networks (RNNs) [RHW86b] and substantial improvements have been subsequently introduced, for example through LSTMs [HS97b]. However, RNNs were still considered difficult to train, as they posed the risk of vanishing or exploding gradients [PMB13], and their sequential nature made parallelization across training samples difficult.

Original Transformer Model. One of the breakthroughs in deep learning for NLP came along with the Transformer model [VSP+17], which proposed a different architecture, without using recurrence, and instead relying on attention mechanisms to learn dependencies in the input data. Loosely speaking, attention [BCB15], and in particular self-attention [VSP+17], is used in deep learning to integrate the information from different parts of a sequence of inputs to create a global representation of that sequence. An attention layer is defined in [VSP+17] as a function that takes a triplet of vectors (Q, K, V) – queries, keys and values, and outputs $f(Q, K, V) = \sigma(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the dimension of Q and K, and σ is the softmax function; more generally, [VSP+17] use a multi-head attention, which consists of multiple stacked attention layers, that can run in parallel. The Transformer model consists of an encoder and a decoder; the encoder takes the input sequence, and maps it to a function containing multi-head attention, fully connected layers, and residual connections. The decoder has a similar structured to the encoder, with an additional sub-layer that performs multi-head attention with the output of the encoder; therefore, the inputs to the decoder are the output at the previous position, along with the encoder of the input. The Transformer model has been successfully used for language translation tasks, and for language modelling, to predict the next word in a sentence.

Transformer Versions. However, one disadvantage of the original Transformers is their fixed-length context, determined by the length of the input sequence, which allows for little connectivity between separate parts of the input data. To solve this challenge, the Transformer-XL [DYY+19] model was introduced, which does not have a sequence-length limit, and which re-uses information from the previous hidden states, to enable learning a larger context. The

experimental results in [DYY+19] show that the Transformer-XL has superior performance on language modelling tasks, being able to generate longer sentences, compared to the original model. Other versions of the original Transformer model have been introduced, such as the BERT [DCLT19] model, which can learn bi-directional deep representations from unlabelled text, in a self-supervised manner; pre-trained BERT models are thus used as popular feature extractors, and their representations can be further finetuned on different tasks.

NLP Datasets. Transformer models can be trained to solve a wide range of NLP tasks, such as machine translation, language modelling or question answering. We briefly describe the datasets used for the NLP experiments presented in this thesis, which focus on language modelling for text generation and question answering. For word-level language modelling, we use in Chapter 3 the Wiki-Text-103 [MXBS16] dataset to train the Transformer-XL model [DYY+19], for predicting the next word in a sentence; this dataset contains text from 28,000 articles, comprising in total of 103 million tokens, with an average of 3,600 tokens per article, and is split into train, validation and test. For question answering tasks, we use in Chapter 5 the SQuAD1.1 dataset [RZLL16], which contains 100,000 pairs of questions and answers; when given a question and a passage from Wikipedia containing the answer, the task is to predict the range where the answer is found within the excerpt.

2.1.4 Generalization in Deep Learning

As discussed in Section 2.1.1, one overarching goal of machine learning is to obtain a small generalization error, defined as a small difference between the test and train error. Different techniques from statistical learning theory [Vap99] have been developed to obtain bounds on the generalization error, many of them based on some measure of complexity of the hypothesis space, such as the Vapnik-Chervonenkis (VC) dimension, Rademacher complexity, or other measures presented in detail in [SSBD14]. While these techniques can be very powerful for simpler models, their usage is limited in the case of neural networks, which have a very complex hypothesis space, capable of perfectly fitting the training set [ZBH+17]; therefore, most of the generalization bounds developed for neural networks [BFT17] can only be used as guidelines to better understand the factors driving their remarkable practical generalization abilities, while non-vacuous generalization bounds have been developed for small models and datasets [DR17].

Insights into neural network generalization. There are several recent works that challenge the traditional view of generalization, when in comes to neural networks. For example, the work of Zhang et al. [ZBH+17] shows that neural networks have the remarkable ability of fitting any input data, including random noise; this implies that the training error and model complexity alone are not enough to infer generalization on unseen data. More recently, the fundamental concept of bias-variance trade-off [HTFF09] from statistical machine learning has also been challenged. The consensus used to be that models with high capacity have low bias and high variance, and are therefore more likely to overfit. However, new work [BHMM19, NKB+21] showed that if the model capacity exceeds an "interpolation threshold" [BHMM19], defined as reaching almost 0 training error, the generalization error of the model decreases; thus, instead of the traditional U-shaped curve for the test error, it is more likely to have a "double descent" curve [BHMM19]. This has been further validated in practice in [NKB+21], where it was shown that double descent does not only occur when increasing the model size, but also when increasing the number of training steps.

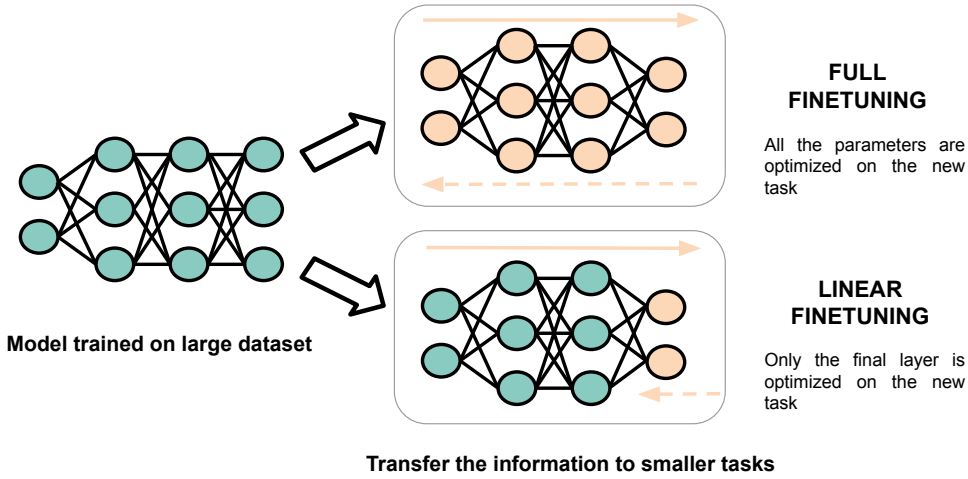


Figure 2.1: Overview of the two most popular methods used for transferring information from large pre-trained models: full and linear finetuning. Dashed lines show the gradient flow via backpropagation.

Flat Minima. Another direction towards understanding generalization in deep learning focuses on the geometry of the loss surface and, in particular, the training dynamics of SGD. It has been hypothesized that "flat" minima have better generalization abilities [HS97a], while the opposite "sharp" minima should be avoided. While the notion of flatness can have different definitions [HS97a, KMN+17, DPBB17], the basic intuition for a one dimensional error curve is that a local minima is flat if there is a wide region around it in which the loss changes very little [DPBB17]. While the notion of flatness has gained a lot of popularity, its definition has also been challenged [DPBB17].

Recent work [FKMN21] has proposed an optimization method aimed at finding solutions with better generalization, by optimizing against the sharpness of the loss function. The method of sharpness-aware minimization [FKMN21] (SAM) has been the inspiration for our proposed method from Chapter 5. We provide a more detailed explanation for SAM in Section 5.2.

2.1.5 Transfer Learning

In a nutshell, transfer learning can be loosely defined by the methods used for leveraging information from existing trained models, unto new, usually less complex, tasks. Thanks to their ability to learn complex representations of the data, deep learning models have been used in transfer learning as feature extractors, for a wide range of applications, from natural language processing to computer vision. It is common to distinguish between two separate setups: (1) *linear finetuning*, or using fixed features (i.e. the parameters in all previous layers are fixed) to train linear models on new tasks; (2) *full finetuning*, or using the pre-trained model as an initialization, to then further optimize its parameters on the downstream task. A visualization of these two setups is presented in Figure 2.1.

For computer vision applications in particular, it was shown that the linear models trained on fixed features extracted from CNNs pretrained on ImageNet could outperform those trained on existing hand-engineered features [DJV+14, SRASC14]. Furthermore, it is well-established that full finetuning generally leads to better results compared to linear finetuning, particularly when the upstream and downstream tasks are very different [KSL19], and also that better models on the upstream task lead to better generalization performance on the downstream tasks.

The concept of transfer learning is widely used also for NLP models and tasks. For example, it has been shown that pre-trained BERT models [DCLT19] can be successfully finetuned on a wide range of language understanding tasks [WSM+18], and also on question answering [RZLL16] or sentence completion tasks [ZBSC18]. In all cases, the pre-trained model is used for full finetuning on the downstream task, without any changes to the original architecture, apart from the output layer. We also use BERT-finetuning on the SQuAD dataset in Chapter 5.

2.2 Fundamentals of Neural Network Pruning

In this section we move to an overview of existing pruning methods, categorizing them over different criteria: the time when pruning is performed, pruning granularity, or the metric used for pruning. A substantially more comprehensive overview of existing pruning methods can be found in [HABN+21], from which we also draw inspiration. Furthermore, we discuss aspects that relate pruning and generalization of neural networks.

2.2.1 Overview

The idea of pruning, i.e. setting to zero a subset of the weights in a neural network, has been known for quite a long time. It was initially proposed as a regularization method [LDS90], since it was believed that neural networks are prone to overfitting, due to their large number of parameters. As we have also discussed in Section 2.1.4, today we know that while neural networks have a massive parameter space, they are not as likely to overfit as previously believed. However, there are still a lot of redundancies in the parameter space of a neural network, as they are capable of perfectly fitting random data [ZBH+17], and it was shown that large, randomly initialized neural networks can contain untrained subnetworks that perform as well as similarly sized trained models [RWK+20]. Therefore, sparsifying neural networks, to take advantage of existing redundancies, for improved efficiency and storage, is a relevant research direction, which might also help uncover new properties of neural networks.

When pruning a neural network, we can set to zero its connections (weights), or entire neurons or filters. The highest granularity is represented by pruning individual weights, known as *unstructured pruning*. We will mostly focus on methods developed for this type of pruning, since it has received the most attention in the sparsity literature. The case when neurons or filters are pruned is known as *structured pruning*; compared to the unstructured case, with structured pruning we can obtain immediate speed-up, since the layers become smaller. However, recent software developments [Dee21] have made it possible to obtain practical speed-ups also for the unstructured case, and there is increased interest in hardware specialized for sparsity [Gra21]. In our presentation, we will also include *semi-structured* pruning methods, which already have hardware support for speed-up [MLP+21].

Some of the most widely-used, successful and, at the same time, simple pruning criteria are those based on *magnitude*. Namely, in the case of unstructured pruning, we set to zero the weights with an absolute value, i.e. magnitude, below a threshold and keep the remaining weights unchanged; the threshold is determined such that a certain sparsity target is obtained. It is common to consider different ways in which the pruned weights are distributed across the network: *global magnitude pruning* computes the pruning threshold from the vector containing the absolute value of the weights across all layers, while *uniform magnitude pruning* prunes each layer individually, at the same sparsity target sparsity level. Other sparsity distributions prune to a higher sparsity the layers with more parameters; for example, the Erdős-Rényi

sparsity distribution [MMS+18] prunes a layer with $n^{\ell-1}$ input neurons and n^{ℓ} output neurons to a level proportional to $\frac{n^{\ell-1}+n^{\ell}}{n^{\ell-1}\cdot n^{\ell}}$, while the Erdős-Rényi-Kernel (ERK) [EGM+20] prunes convolutional layers with dimension $n^{\ell-1} \times n^{\ell} \times w^{\ell} \times h^{\ell}$ to a sparsity level proportional to $\frac{n^{\ell-1}+n^{\ell}+w^{\ell}+h^{\ell}}{n^{\ell-1}\cdot n^{\ell}\cdot w^{\ell}\cdot h^{\ell}}$. For all pruning distributions, only the weights of the convolutional or fully connected layers are considered for pruning, while biases and normalization parameters are kept dense. In the case of uniform pruning, usually the first and last layers are not pruned, since they can have a disproportionate effect on model performance, without contributing significantly to the overall speed-up. Similarly, magnitude pruning can be defined for structured pruning, where we can consider the ℓ_1 norm of each component (e.g. neuron or filter) and remove (set to zero) the smallest ones.

2.2.2 Unstructured Pruning Methods

In this section, we provide a taxonomy of methods for unstructured pruning, based on the time during training when pruning is performed. We re-use this classification in Chapter 4, and also partially in Chapters 3 and 5. A rough categorization considers the case when pruning is performed after or during training. We also discuss hybrid methods that contain elements from both categories. Lastly, we provide a discussion on obtaining practical speed-up from unstructured sparsity.

Progressive Sparsification Methods

Some of the most popular and successful methods for pruning start from a fully trained dense model, to which pruning is applied progressively, with a gradual increase in the sparsity level, until a target sparsity level is reached [HPTD15, SA20, FKA21]. Between each two progressive pruning steps, the remaining weights are finetuned for a small number of epochs. Furthermore, at the end of the pruning schedule, it is common to finetune the resulting sparse model for an extended number of epochs, which further boosts its prediction accuracy. We call methods following this training and pruning pattern *progressive sparsification methods*. Typically, the sparsity level is increased following a polynomial schedule, the most common choice being a third degree polynomial [ZG17].

Gradual Magnitude Pruning. The most popular method in this category, which also achieves close to state-of-the-art results on image classification [GEH19] or language modelling tasks [KA22], prunes the weights based on their magnitude, and it is commonly known as *Gradual Magnitude Pruning* (GMP), with different choices for the sparsity distribution (e.g. uniform or global).

Hessian-Based Pruning. One natural question we could ask is whether there exist better pruning criteria than using the magnitude of the weights. Early work [LDS90, HSW93] explored the idea of removing the weights with the least impact on the loss function. We assume for simplicity that we want to remove one weight at a time; namely, for model parameters $\boldsymbol{\theta} \in \mathbb{R}^N$ and loss function $L(\boldsymbol{\theta})$, we want to find the perturbation $\boldsymbol{\delta} \in \mathbb{R}^N$ with $\boldsymbol{\theta}_q + \boldsymbol{e}_q^T\boldsymbol{\delta} = 0$ (where $\boldsymbol{e}_q$ is the standard basis vector with 1 on the q-th component), which minimizes:

$$\min_{1\leq q\leq N} \min_{\boldsymbol{\delta}\in\mathbb{R}^N} L(\boldsymbol{\theta}+\boldsymbol{\delta}) - L(\boldsymbol{\theta}), \quad \text{s.t} \quad \boldsymbol{\theta}_q + \boldsymbol{e}_q^T \cdot \boldsymbol{\delta} = 0$$

For a fixed q, we can use a second-order Taylor approximation to obtain $L(\boldsymbol{\theta}+\boldsymbol{\delta}) \approx L(\boldsymbol{\theta}) + \nabla L(\boldsymbol{\theta})^T\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^T H_{\boldsymbol{\theta}}\boldsymbol{\delta}$, where $H_{\boldsymbol{\theta}}$ is the Hessian of L computed for $\boldsymbol{\theta}$. Since the model

$\boldsymbol{\theta}$ is assumed to be trained until convergence, we have $\nabla L(\boldsymbol{\theta}) \approx 0$. Using Lagrange multipliers, the initial optimization problem can be reduced to:

$$\min_{\boldsymbol{\delta},\lambda} \frac{1}{2}\boldsymbol{\delta}^T H_{\boldsymbol{\theta}} \boldsymbol{\delta} + \lambda(\boldsymbol{\theta}_q + \boldsymbol{e}_q^T \boldsymbol{\delta}),$$

for which we obtain $\lambda = \frac{\boldsymbol{\theta}_q}{[H_{\boldsymbol{\theta}}^{-1}]_{qq}}$ and $\boldsymbol{\delta} = -\frac{\boldsymbol{\theta}_q H_{\boldsymbol{\theta}}^{-1} \boldsymbol{e}_q}{[H_{\boldsymbol{\theta}}^{-1}]_{qq}}$. Moreover, the loss perturbation $L(\boldsymbol{\theta} + \boldsymbol{\delta}) - L(\boldsymbol{\theta})$ can be approximated by $\boldsymbol{\delta} L_q = \frac{\boldsymbol{\theta}_q^2}{2[H_{\boldsymbol{\theta}}^{-1}]_{qq}}$. Therefore, to find which weight can be set to zero, and how to update the model to obtain the minimal loss perturbation, we sort the values $\boldsymbol{\delta} L_q$, pick the component q corresponding to the smallest one, compute $\boldsymbol{\delta}$ and then update the weights by $\boldsymbol{\theta} + \boldsymbol{\delta}$. This technique is known as optimal brain surgeon (OBS) [HSW93], while the case then the Hessian is assumed to be diagonal is known as optimal brain damage (OBD) [LDS90].

One obvious problem with the above approach is the assumption that we have access to the inverse of the Hessian matrix. In the case of neural networks with millions of parameters, simply computing the Hessian matrix is a daunting task, let alone inverting it. One approach towards solving this issue was initially proposed in [HSW93], by approximating the Hessian with the empirical Fisher information matrix (FIM). The empirical FIM, defined as $\hat{F}_{\boldsymbol{\theta}} = \frac{1}{n}\sum_{i=1}^{n}[\nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(x_i), y_i) \cdot \nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(x_i), y_i)^T]$, where the average is taken over the training set $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$ is in fact an approximation of the true FIM, defined as $F_{\boldsymbol{\theta}} = \mathbb{E}_{x,y\sim\mathcal{D}}[\nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(x), y) \cdot \nabla_{\boldsymbol{\theta}} \ell(h_{\boldsymbol{\theta}}(x), y)^T]$, where the expectation is taken over the input distribution; the FIM is known to be equal to the average Hessian over the data distribution. The authors of [SA20] take advantage of this approximation, together with the fact that the empirical FIM is a sum of rank-one matrices for which inversion is facilitated by the Woodbury or Sherman-Morrison lemmas, and compute the inverse of the empirical FIM. They further assume a block-diagonal structure of the matrix, and use gradients computed over batches of samples, for a faster approximation. To deal with multiple weights removal, they only keep the weights with the largest statistics $\boldsymbol{\delta} L_q$, ignore any correlations by simply summing the updates $-\frac{\boldsymbol{\theta}_q H_{\boldsymbol{\theta}}^{-1} \boldsymbol{e}_q}{[H_{\boldsymbol{\theta}}^{-1}]_{qq}}$, followed by applying the corresponding mask to ensure sparsity. The pruning method introduced in [SA20] is called *WoodFisher*, and it can be plugged-in to the progressive sparsification scheme, in place of the magnitude pruning criterion, to obtain better performance than GMP, particularly at higher sparsity levels.

Improvements for this algorithm, in terms of implementation efficiency and memory costs, have been proposed in the *M-FAC* method [FKA21]. In particular, the authors show that the dot product between the inverse empirical FIM with any vector can be computed efficiently, by un-rolling the recurrence resulted from the Sherman-Morrison lemma; this approach eliminates the need to store the entire FIM inverse, and also does not make any additional assumptions on the structure of the matrix (e.g. block-diagonality). Both M-FAC and WoodFisher provide better results, in terms of pruning accuracy, compared to GMP, and we will use them as baselines for our methods in the following chapters.

Regularization Methods

While progressive sparsification methods are very successful at obtaining sparse models with a small accuracy drop compared to the baseline, they have important drawbacks: any speed-ups obtained from pruning can only be observed at inference time, and they typically take a longer time to train, since they assume training the dense model first. For this reason, a

substantial amount of research interest has been devoted in recent years towards training sparse models *from scratch*, by employing different sparsity-promoting mechanisms. While the literature in this area is vast, we will only provide a brief overview which includes the methods we use for comparison in later chapters. Furthermore, we will use the umbrella term of *regularization methods*, despite the fact that many of these methods induce sparsity through very different mechanisms, such as explicit regularization [LWK18, YWL20], reparameterization of the weights [MAV17, KRS+20], gradient feedback [LSB+20] or by a continuous pruning and regrowth of the weights [EGM+20].

Sparse Training. One interesting line of work is to obtain accurate sparse models, while keeping a fixed sparsity during training, both during the forward and backward passes; this is known as *sparse training*. One method achieving close to state-of-the-art results with this approach is RigL (Rigging the Lottery) [EGM+20]. While training is always performed in the sparse support, at a constant sparsity level, the masks are updated periodically, by removing some of the existing connections, are replacing them with connections for which the gradients have the highest magnitude; to achieve this, the full gradient is computed every few hundred iterations. This simple prune-and-regrowth scheme has the potential for training time speed-up, and it has shown promising results, particularly when training is performed for an extended time; in this regime, 90% sparse RigL models can reach similar performance to the dense baseline, when using a ERK sparsity distribution. However, when trained for the same number of epochs as the standard baseline, the results obtained with RigL are still well below those obtained with progressive sparsification methods. In Section 3.2 from Chapter 3 we provide more details on earlier sparse training methods, as well as on other methods [JPR+20] that perform similar or better than RigL.

Training Sparse Models from Scratch. We now discuss regularization methods that still train sparse networks from scratch, although they do not maintain a constant high sparsity level during both forward and backward passes, throughout training. Typically, with these methods the sparsity is increased, until a desired target level is reached; notably, there exist variants of GMP that do not start from a fully trained dense model, such as [GEH19], and fit into the current category, but these typically suffer a loss in accuracy, compared to the corresponding progressive sparsification methods. One example of a method for training sparse models from scratch is Dynamic Model Pruning with Feedback (DPF) where the magnitude pruning masks are updated via model feedback; namely, the model always has access to the dense weights, which are optimized with SGD, using the full gradient computed on the masked weights. The update can be written as $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \nabla L(\tilde{\boldsymbol{\theta}}_t)$, where $\tilde{\boldsymbol{\theta}}_t$ are the masked parameters.

Another sparse regularization method to show promising results proposed a sparse reparameterization of the weights [KRS+20], through the use of the soft thresholding operator, roughly defined as $\tilde{\boldsymbol{\theta}} = \text{sign}(\boldsymbol{\theta}) \cdot \sigma(|\boldsymbol{\theta}| - \alpha)$, where σ is the ReLU function, and α is the threshold. In addition to the weights, the proposed method STR [KRS+20] also learns individual thresholds per layer, modelled though a sigmoid function. This is motivated by one the main goals of the method, which was to learn an optimal sparsity distribution, in terms of both prediction accuracy and floating point operations (FLOPs). The sparsity level in STR is gradually increased, but the method starts from a randomly initialized model. While STR can achieve good results, one of its main drawbacks is that it heavily relies on hyperparameter tuning, and in particular the target sparsity level is greatly influenced by the weight decay parameter. We use STR as a baseline in Chapters 3 and 5, and we also study its generalization properties in

Chapter 4. More recently, another method proposing soft thresholding and a different sparsity distribution per layer has been shown to achieve very good results on image classification tasks [VDV23].

Explicit Regularization. We now briefly discuss a category of pruning methods that obtain sparsity by integrating different types of explicit regularizers in the training objective. One of the most well-known sparsity-inducing regularizers is the ℓ_1 norm, or Lasso [Tib96], which has also been used for pruning [LWF+15]; one of its disadvantages, however, is that it is not scale invariant, which makes it difficult to use it for higher sparsity targets. Another obvious choice for sparsity regularization is ℓ_0, which explicitly restricts the number of non-zero elements; one problem with ℓ_0, however, is that it is non-differentiable, and cannot be used to compute gradients. There have been pruning methods [LWK18] that provide a smooth approximation of the ℓ_0 objective, which allows optimization via SGD; we also use the ℓ_0 method proposed in [LWK18] as a baseline when pruning language models in Chapter 5. Furthermore, the ℓ_0 objective can be implicitly integrated into the optimization objective via iterative hard thresholding (IHT) methods [BD08], which we discuss in detail in Chapter 3. Another sparsity-inducing regularizer that is differentiable almost everywhere is the Hoyer regularizer [Hoy04], which was also recently used for pruning neural networks in [YWL20].

Variational Methods. A different approach to weight pruning considers a Bayesian treatment of the training objective. This is based on the observation [KSW15] that training a model using Gaussian dropout noise [SHK+14] is equivalent to optimizing a Gaussian variational posterior of the parameters, assuming a log-uniform prior over the weights; this framework is known as variational dropout. The authors of [MAV17] later proposed *Sparse Variational Dropout* (SparseVD), and showed that under the setup of [KSW15] it is possible to train individual dropout rates for each weight. Intuitively, weights with large dropout rates are not important for the training objective, and thus can be safely removed after training. Indeed, the authors of [MAV17] showed that models trained with SparseVD can be pruned in a single step, by removing the weights with variance above a threshold, with no drop in accuracy. Moreover, the framework can be extended to structured sparsity [NMAV17, LUW17]. Subsequently, the authors of [GEH19] showed that SparseVD can be used also for large-scale models. One drawback, however, is that training is slower because there are two times more parameters, and also the model is very sensible to the choice of hyperparameters.

Lottery Ticket Hypothesis

We now discuss an orthogonal research direction in neural network pruning, namely the *lottery ticket hypothesis* (LTH), which we will also revisit in Section 2.2.4, as it has potential implications for better understanding the generalization properties of (sparse) neural networks. The seminal work which introduced the LTH [FC19] argues against the previously accepted belief that training sparse neural networks from scratch cannot outperform finetuning an already pruned model. Namely, the authors of [FC19] formulate the LTH as follows: there exists a subnetwork of a dense neural network, such that, when trained in isolation starting from a specific initialization, the subnetwork can match the test accuracy of the original dense model; furthermore, the subnetwork reaches this accuracy faster than the original model, in terms of number of training iterations. This hypothesis would have both practical and theoretical implications: on one hand it would motivate more research into finding good subnetworks at initialization, for which training in isolation would substantially reduce computational costs;

furthermore, it could lead to a better understanding of the theoretical study on optimization and generalization of neural networks.

The authors propose a practical algorithm for finding such subnetworks, called *winning tickets*, following a repeated training and pruning scheme called *iterative magnitude pruning* (IMP). Namely, the IMP scheme consists of the following consecutive steps, reproduced from [FC19]:

1. start from a randomly initialized neural network $h(\boldsymbol{\theta}_0 \odot m)$ (the mask m is initialized as $\mathbb{1}^N$ for the first iteration)
2. train h for t iterations, obtaining parameters $\boldsymbol{\theta}_t \odot m$
3. prune $s\%$ of the parameters $\boldsymbol{\theta}_t \odot m$, and update the mask m
4. reset the remaining parameters in $\boldsymbol{\theta}_t \odot m$ to their initial values from $\boldsymbol{\theta}_0$, to obtain the winning ticket $h(\boldsymbol{\theta}_0 \odot m)$
5. if the sparsity target S is not reached, repeat from the first step

Moreover, the authors of [FC19] emphasize the importance of the initialization of the subnetworks, which has to match the initialization of the dense model; otherwise, the subnetwork underperforms the original baseline. In terms of results, the original paper shows that the LTH holds for small fully connected and convolutional networks trained on MNIST [LBBH98] and CIFAR-10 [KH+09] datasets. However, following works [GEH19, LSZ+19] were not able to validate the hypothesis for large-scale models and datasets, e.g. on ImageNet. Subsequently, it was shown that it is possible to obtain sparse models that match the original accuracy when trained in isolation by *weight rewinding* [FDRC20]; namely, in the IMP algorithm, the surviving weights after pruning should not be reverted to their initialization, but rather to the values at an early epoch during dense training. While LTH remains an interesting direction that has seen a lot of research interest since its definition, it is unclear whether it can be used as a pruning method, since the IMP algorithm assumes training until convergence for each iteration, which can be very costly for large models and datasets. We further note that extensions of LTH have been studied for transfer learning [CFC+21], which we discuss in Chapter 4, and also for language models [CFC+20].

Post-Training Pruning

We now explore a different direction in neural network pruning, which aims at sparsifying trained dense models without any additional finetuning. This also implies that the training data, except for a small subset used for calibration, is no longer needed for obtaining the sparse model. Such methods can be categorized under the umbrella term *post-training pruning*, and are of great practical importance, as they would facilitate the rapid deployment of sparse models on to smaller devices; this is particularly useful in cases where the dense model is so large (e.g. the GPT models [BMR+20]), that any amount of finetuning would be extremely costly.

One of the early methods for post-training pruning was AdaPrune [HCI+21], which proposed optimizing the pruned weights to minimize the distance in the layer output between the sparse and dense model. Namely, for a layer ℓ, with weights $\boldsymbol{\theta}_\ell$ and layer input $\boldsymbol{x}_\ell$, the proposed objective is $\min_{\boldsymbol{\theta}'} \|\boldsymbol{\theta}_\ell \boldsymbol{x}_\ell - \boldsymbol{\theta}' \boldsymbol{x}_\ell\|^2$, where $\boldsymbol{\theta}'$ are the sparse weights for the respective layer. Optimization is performed over a small calibration set consisting of randomly sampled training

points. Following work [FA22b] proposed global AdaPrune, which optimizes the above objective globally across all layers, to take into account the compounding errors which are ignored when performing optimization independently across layers.

Using the distance between layer outputs as the loss function, the optimal brain compression (OBC) [FA22a] finds the optimal weights to prune, by using an efficient implementation of the optimal brain surgeon framework [HSW93]. The OBC method obtains state-of-the-art results for post-training pruning on both computer vision and language models, and can further be adapted to post-training quantization, also at very large scale [FAHA23]. Furthermore, the framework introduced in [FA22a] can be further improved [FA23] in terms of efficiency to make it suitable for post-training pruning of massive language models, such as GPT-3 [BMR+20], for which the authors [FA23] showed that it is possible to prune 50-60% of the weights without retraining, and with almost no impact on model quality.

Practical Speed-Up from Unstructured Sparsity

We now briefly discuss some applications where sparsity can be extremely useful. One of them is to obtain faster inference for deep learning models. Recently, specialized software for CPU-based inference has been proposed [KKG+20, Dee21], enabling substantial speed-up from sparsity, including from unstructured sparsity patterns. Notably, inference forms an important fraction of the total energy usage for deep learning models [PGH+22], and speed-ups on CPUs are particularly important for edge devices, which often lack hardware accelerators. For example, sparsity can enable on edge devices local predictions for deep learning models used in certain applications (e.g. translation, object recognition); performing inference locally is useful in situations where data cannot be sent to the cloud, due to, for example, poor internet connection or privacy concerns. Additionally, sparse deep learning models can be finetuned to smaller tasks, even on edge devices. We showcase an example in Chapter 4, where we fix the sparse backbone of a pretrained model and adapt only the final classification layer to the new tasks. With this procedure, we are able to obtain two to four times faster training on a CPU similar to the requirements of a laptop; while the tasks we consider in Chapter 4 might not correspond exactly to tasks used in real-life applications, they can still be representative, in terms of size and complexity.

Another important application of sparsity would be to accelerate the training time of deep learning models. This would also mean adapting the backpropagation algorithm used for computing the gradients of the parameters to handle sparsity. This has been done recently [NPI+23], where the authors showed speed-up during backward passes on CPUs. However, to enable the end-to-end faster training of a sparse model, such an implementation would need to be efficiently adapted to hardware accelerators. Furthermore, this would potentially need to be accompanied by specialized hardware for sparsity, for which there is an active interest [Gra21]. Until real time speed-up can be achieved from training sparse neural networks, it is very common in the pruning literature [EGM+20, JPR+20, SA20] to estimate the speed-up theoretically, for example by measuring the number of floating point operations (FLOPs); we also adopt this approach in Chapter 3.

2.2.3 Structured and Semi-Structured Pruning

We now provide a brief overview of structured and semi-structured pruning methods. As we have seen in the previous section, unstructured pruning has received by far the most research interest, with a large number of methods developed for different training scenarios. However, it is argued

that unstructured pruning has limited practical advantages, as it is difficult to obtain acceleration from unstructured sparse matrices generated by these methods [WWW+16, MLY+21], and the expected speed-up is mostly theoretical. Therefore, a more principled approach to pruning is often proposed, namely *structured pruning*, in which entire neurons or filters are sparse; this would effectively reduce the size of the layers, from which one can obtain immediate practical acceleration, due to the smaller size of the network.

Several methods have been developed for structured sparsity; for example, [WWW+16] propose training models with a group Lasso penalty term, to enforce group sparsity, while [WGFZ19] propose a new weight reparameterization, to which they apply Hessian-based structured pruning, inspired by the OBD and OBS derivations [LDS90, HSW93]. Furthermore, [LAJ19] define a budget-aware objective, and learn the parameters of dropout tensors applied to the outputs of each layer, while also leveraging knowledge distillation [HVD15]. Along the lines of parameterizing dropout, using a variational objective, several works inspired from [KSW15, MAV17] have been developed for structured pruning, such as [NMAV17, LUW17]. A new approach for structured pruning using an actor-critic algorithm from reinforcement learning, to learn optimal pruning ratios per layer has been developed in [HLL+18]. Moreover, a different line of work proposes methods to learn multiple networks, at different computational budgets, at once, to allow for rapid deployment to edge devices [CGW+19, YJL+20, YYX+19], but we discuss these methods in more detail in Section 5.2 from Chapter 5.

While it is true that structured pruning can leverage practical acceleration by effectively reducing the size of the model, we note that it also allows for lower sparsity, compared to unstructured pruning, and typically the performance in terms of accuracy can be substantially affected. Moreover, because structured pruning alters the dimensions of the layers, this can introduce many difficulties, for example with residual connections or when performing pruning globally [TGL+22]. We also emphasize that practical speed-ups can be obtained with unstructured pruning, thanks to specialized software [KKG+20, Dee21], and we show this in Chapters 3 and 4.

Semi-Structured Sparsity. We now focus our attention on a new pruning pattern, which is less fine-grained than unstructured pruning, but allows for more flexibility compared to structured pruning. Namely, we discuss the $N : M$ sparsity pattern [MLP+21, ZMZ+21], where for each contiguous group of M weights, N have to be zero. The original $2 : 4$ pattern was proposed by [MLP+21] and it has been shown to support inference acceleration on NVIDIA Ampere GPUs [Nvi20]. The initial recipe for obtaining $2 : 4$ sparse models proposed in [MLP+21] was to train a dense model to convergence, prune one-shot to $2 : 4$ sparsity using magnitude pruning, and then do sparse finetuning to recover the accuracy of the pruned model. While the results showed that with this training method one could obtain sparse models that do not lose accuracy w.r.t. the dense baseline, it is still considered costly since it relies on a fully trained dense model, similar to the methods we discussed for unstructured pruning using progressive sparsification.

Following work [ZMZ+21] generalize the concept to $N : M$ patterns, with $4 : 8$ being the next most popular, and proposed a method for training such models from scratch, without a dense baseline. Namely, the authors proposed a new type of straight-through gradient estimator [BLC13] which was less likely to introduce large errors in the gradient, and optimized the parameters of the dense model, using the estimation of the gradient computed on the sparse model. The $N : M$ sparse models trained with this method maintained accuracy w.r.t. the dense baseline, for both $2 : 4$ and $4 : 8$ patterns. Subsequently, [HCI+21] proposed a

method for finding masks such that both the weight matrix and its transpose follows the $N:M$ pattern, such that they can be used for both forward and backward passes, to obtain training time speed-up. Furthermore, [ZLL+22] proposed a novel method for finding the best weights to prune to preserve the $N:M$ pattern, while [PY21] proposed a method based on channel permutations to improve the performance of $N:M$ sparse models. Based on these new results, we can therefore conclude that the area of semi-structured sparsity is an active one, which provides exciting prospects for accurate sparse models with practical speed-ups. We also show that the methods we propose can be extended to $N:M$ pruning, particularly $2:4$ and $4:8$, in Chapters 3 and 5.

2.2.4 Generalization Properties of Sparse Models

The previous sections were meant to offer a glimpse into the vast number of methods developed for pruning neural networks. Despite the great research interest in this field, most works focused on proposing or improving methods for pruning, for better accuracy and speed-up, and there is still not enough known about the effects of pruning on model generalization. The early work on optimal brain surgeon (OBS) [HSW93] motivated the need for pruning to improve a model's generalization, since overparameterized networks were believed to be more prone to overfitting. This intuition is rooted in the well-known bias-variance trade-off in statistical machine learning. However, as we have previously discussed in Section 2.1.4, this traditional view on generalization does not seem to hold for deep learning models, which, despite their massive capacity, are still able to generalize very well. Nonetheless, moderately sparse models have been shown to improve generalization w.r.t. to their dense counterparts [HPTD15, FC19, EGM+20, HABN+21], while temporarily removing a small fraction of the weights can give a performance boost to the dense models [HPN+17]. These observations motivate the need to understand how does pruning influence generalization in deep learning?

Generalization Guarantees for Compressed Models. Going back to the earlier connection between generalization and compression (and pruning, in particular), several works [AGNZ18, ZVA+19] have focused on obtaining generalization bounds for compressed models. For example, [ZVA+19] proves non-vacuous PAC-Bayes bounds [Cat07, McA03] for existing pruning and quantization methods, at ImageNet scale. In [AGNZ18], the authors propose a compression scheme based on random matrix projection, which controls the perturbations from the original parameters; the compression algorithm is motivated by the empirical observation that noise injected into the input to a layer rapidly dissipates as it reaches higher layers. Moreover, the theoretical framework in [AGNZ18] is general enough to enable deriving generalization guarantees for subsequent pruning methods [BLG+19]. Specifically, the authors of [BLG+19] propose a pruning scheme that uses a subset of the data to estimate a measure of sensitivity or importance for each weight, while at the same guaranteeing that the output of the resulting sparse model is close to that of the original dense baseline. This framework is later extended in [BLG+22], where the authors show that the pruning scheme achieves good results in the iterative pruning and retraining setup [RFC20], or when pruning at initialization and finetuning [LAT19]. Similar guarantees are also presented in [LBL+20], where the framework in [BLG+19] is extended for filter pruning.

Exploring Redundancies in a Neural Network. While over-parameterization in a neural networks seems to benefit generalization [BHMM19], several works have pointed out that neural networks contain many redundancies in their parameters. For example, [DSD+13]

showed that it is possible to predict the weight of a model only from a learned subset of 5% of the total parameters, while [RWK+20] showed that, surprisingly, a large network at random initialization contains a subnetwork that matches the accuracy of a larger model trained on ImageNet. These redundancies have also been explored by the Lottery Ticket Hypothesis (LTH), which we described in detail in Section 2.2.1, and which showed that sparse networks can be trained in isolation to match the accuracy of the dense baseline. Following work [ZLLY19] confirmed that matching the initializations between the subnetwork and the dense model is important to obtain winning tickets on smaller datasets and models, but also showed that matching only the sign of the weights at initialization can lead to winning tickets. Moreover, they showed the existence of "supermasks", which applied to a random initialized model resulted in substantially higher accuracy than for a random mask. Despite these documented redundancies, it seems that they cannot be easily discarded; for example, [EGM+20] showed that pruning a larger model gives better results compared to training a small model from scratch, with the same number of parameters. Moreover, further scaling the original model and pruning it to match the original number of parameters can give a substantial boost in accuracy.

Pruning Factors Influencing Generalization. Several works have focused on analyzing in detail how pruning can influence generalization, by studying its impact on individual training samples [JCR+22] or by relating the pruning stability to flatness of the loss function [BMBE20]. Specifically, [JCR+22] use the definition for the influence of a data point on the training process proposed in [PGD21], which measure the ℓ_2 distance between the predicted probabilities and one-hot output vector early during training. With this metric, [JCR+22] showed that, compared to the dense baseline, pruning can improve learning of the more influential samples, by decreasing their loss, which suggests that pruning can overall lead to better training of the model. They additionally test the effects of random label noise in the data, and find that samples with random labels are more difficult to learn, i.e. have higher loss, compared to those with clean labels, which suggests that pruning has a regularizing effect on the model. Also, sparse models tend to generalize better in the presence of noisy training data, compared to the dense model. While interesting, we note that these conclusions are obtained from models pruned using iterative magnitude pruning (IMP) with learning rate rewinding [RFC20], which trains models at each intermediate sparsity level from scratch, for a much higher number of total training epochs, compared to the baseline. The authors of [JCR+22] address this by performing extended training on the dense model, using cyclical learning rate schedules. However, it would be interesting to see how these findings generalize to other pruning methods, for example sparse training. In Chapter 3 we also show the effects of pruning learning random labels, by directly examining this behaviour on a method that co-trains sparse and dense models.

A different work [BMBE20] argues that pruning stability is related to generalization. In particular, the authors defined pruning instability as the drop in test accuracy before and after a pruning step, and conclude that less pruning stability may in fact, surprisingly, lead to flatter models. Similar to [JCR+22], the pruning methods studied in [BMBE20] sparsify gradually and would therefore be interesting to see how these findings would generalize to other methods. In Chapter 5 we also analyze a method for training prunable models that takes into account pruning stability and is inspired from sharpness-aware minimization [FKMN21].

The Effects of Pruning beyond Accuracy. Another important aspect related to the generalization abilities of sparse neural networks takes into account the performance of these

models on metrics beyond test accuracy. The motivation for this study is given by the fact that pruning methods are typically optimized for achieving similar or better predictive performance compared to the dense baseline, as well as for speed-up. However, these compressed models are meant to be deployed to edge devices, where they can encounter different data distributions, and their behavior in this setting is not clear. Recent works [LBC+21, HCC+19, HMC+20] have pointed out that while pruned models can be indistinguishable from their dense counterparts in terms of accuracy, they can be very brittle when faced with out-of-distribution data [LBC+21], and they can have a disproportionate effect on under-represented subgroups of the data [HCC+19], while potentially exacerbating bias, and implicitly, being less fair than the corresponding dense models [HMC+20]. While these works have analyzed progressive sparsification strategies, [DBC+21] argues that, on the contrary, lottery tickets can in fact have improved robustness, compared to the dense model, while maintaining test accuracy. Tangential to this line of work, we also analyze the behavior of different pruning methods when performing fine-tuning on different downstream tasks, in Chapter 4.

CHAPTER 3

Co-Training Sparse and Dense Models by Alternating Compressed and Decompressed Phases

As previously discussed in Chapter 2, neural network pruning has become one of the most popular approaches to compressing large models. While pruning methods can provide accurate models with smaller memory footprint compared to their dense counterparts, there is still the challenge of reducing the training costs associated with obtaining the sparse models. Moreover, while sparse models can be very close, in terms of accuracy, to the dense ones, it is less understood what are the differences, in terms of generalization and per-sample prediction, between sparse and dense models.

In this chapter, we present a pruning method that attempts to approach these challenges. Namely, we introduce a method that can co-train sparse and dense models, which we call Alternating Compressed/DeCompressed Training (AC/DC). With AC/DC, we can train accurate sparse models at a lower cost compared to the dense baseline, and we are also able to show theoretical guarantees in terms of convergence of our method. Moreover, since AC/DC co-trains sparse and dense models, we can analyze more easily the differences between them, which in turn can facilitate better interpretability of sparse models.

3.1 Motivation and Outlook

Progressive sparsification methods, which were discussed in detail in Section 2.2.2 from Chapter 2, can result in very accurate sparse models, but they require a fully-trained dense variant of the model, from which weights are subsequently removed. A shortcoming of this approach is the fact that the memory and computational savings from compression are only available for *inference*, and not during training itself. This distinction becomes important especially for large-scale models, which can have millions or even billions of parameters, and for which fully-dense training can have high computational and even non-trivial environmental costs [SGM20].

One approach to address this issue is *sparse training*, which essentially aims to remove connections from the neural network as early as possible during training, while still matching, or at least approximating, the accuracy of the fully-dense model. For example, the RigL

technique [EGM+20] randomly removes a large fraction of connections early in training, and then proceeds to optimize over the sparse support, providing savings due to sparse back-propagation. Periodically, the method re-introduces some of the weights during the training process, based on a combination of heuristics, which requires taking full gradients. This work, as well as many recent sparse training approaches [BKML18, MMS+18, EGM+20, JPR+20], which we cover in detail in the next section, have shown empirically that non-trivial computational savings, usually measured in theoretical FLOPs, can be obtained using sparse training, and that the optimization process can be fairly robust to sparsification of the support.

At the same time, this line of work still leaves intriguing open questions. The first is *theoretical*: to our knowledge, none of the methods optimizing over sparse support, and hence providing training speed-up, have been shown to have convergence guarantees. The second is *practical*, and concerns a deeper understanding of the relationship between the densely-trained model, and the sparsely-trained one. Specifically, (1) most existing sparse training methods still leave a non-negligible accuracy gap, relative to dense training, or even post-training sparsification; and (2) most existing work on sparsity requires substantial changes to the training flow, and focuses on maximizing global accuracy metrics; thus, we lack understanding when it comes to *co-training* sparse and dense models, as well as with respect to correlations between sparse and dense models *at the level of individual predictions*.

In this chapter, we take a step towards addressing these questions. We investigate a general hybrid approach for sparse training of neural networks, which we call *Alternating Compressed / DeCompressed (AC/DC)* training. AC/DC performs *co-training of sparse and dense models*, and can return both an accurate *sparse* model, and a *dense* model, which can recover the dense baseline accuracy via fine-tuning. We show that a variant of AC/DC ensures convergence for general non-convex, but smooth objectives, under analytic assumptions. Extensive experimental results show that it provides state-of-the-art accuracy compared to sparse training techniques at comparable training budgets, and can even outperform *post-training* sparsification approaches when applied at high sparsities.

AC/DC builds on the classic *iterative hard thresholding (IHT)* family of methods for sparse recovery [BD08]. As the name suggests, AC/DC works by alternating the standard *dense* training phases with *sparse* phases where optimization is performed exclusively over a fixed *sparse support*, and a subset of the weights and their gradients are fixed at zero, leading to computational savings. (This is in contrast to *error feedback* algorithms, e.g. [CHS+16, LSB+20] which require computing fully-dense gradients, even though the weights themselves may be sparse.) The process uses the same hyper-parameters, including the number of epochs, as regular training, and the frequency and length of the phases can be safely set to standard values, e.g. 5–10 epochs. We ensure that training ends on a *sparse* phase, and return the resulting *sparse* model, as well as the last *dense* model obtained at the end of a *dense* phase. This dense model may be additionally fine-tuned for a short period, leading to a more accurate *dense-finetuned* model, which we usually find to match the accuracy of the *dense baseline*, when moderate sparsity (up to 80-90%) is used in the AC/DC training flow.

We emphasize that algorithms alternating sparse and dense training phases for deep neural networks have been previously investigated [JYFY16, HPN+17], but with the different goal of using sparsity as a regularizer to obtain *more accurate dense models*. Relative to these works, our goals are two-fold: we aim to produce highly-accurate, highly-sparse models, but also to maximize the fraction of training time for which optimization is performed over a sparse support, leading to computational savings. Further, we are the first to provide convergence guarantees for variants of this approach.

We perform an extensive empirical investigation, showing that AC/DC provides consistently good results on a wide range of models and tasks (ResNet [HZRS16] and MobileNets [HZC+17] on the ImageNet [RDS+15] dataset, and Transformers [VSP+17, DYY+19] on WikiText [MXBS16]), under standard values of the training hyper-parameters. Specifically, when executed on the same number of training epochs, our method outperforms all previous sparse training methods in terms of the accuracy of the resulting sparse model, often by significant margins. This comes at the cost of slightly higher theoretical computational cost relative to prior sparse training methods, although AC/DC usually reduces training FLOPs to 45–65% of the dense baseline. AC/DC is also close to the accuracy of state-of-the-art progressive sparsification methods [KRS+20, SA20] at medium sparsities (80% and 90%); surprisingly, it *outperforms* them in terms of accuracy, at higher sparsities. In addition, AC/DC is flexible with respect to the structure of the "sparse projection" applied at each compressed step: we illustrate this by obtaining *semi-structured* pruned models using the 2:4 sparsity pattern efficiently supported by new NVIDIA hardware [MLP+21]. Further, we show that the resulting sparse models can provide substantial real-world speedups for deep neural networks inference on CPUs [Dee21]. We additionally show that the accuracy of the sparse AC/DC models improves with extended training; namely, we are able to obtain 80% and 90% sparse models that outperform the performance of dense models.

An interesting feature of AC/DC is that it allows for accurate dense/sparse co-training of models. Specifically, at medium sparsity levels (80% and 90%), the method allows the co-trained dense model to recover the dense baseline accuracy via a short fine-tuning period. In addition, dense/sparse co-training provides us with a lens into the training dynamics, in particular relative to the sample-level accuracy of the two models, but also in terms of the dynamics of the sparsity masks. Specifically, we observe that co-trained sparse/dense pairs have higher sample-level agreement than sparse/dense pairs obtained via progressive sparsification methods (e.g. GMP), and that weight masks still change later in training.

Additionally, we probe the accuracy differences between sparse and dense models, by examining their "memorization" capacity [ZBH+17]. For this, we perform dense/sparse co-training in a setting where a small number of training samples have *corrupted labels*, and examine how these samples are classified during dense and sparse phases, respectively. We observe that the sparse model is less able to "memorize" the corrupted labels, and instead often classifies the corrupted samples to their *true* (correct) class. By contrast, during dense phases the model can easily "memorize" the corrupted labels. We provide an illustration of this phenomenon in Figure 3.4. This suggests that one reason for the higher accuracy of dense models is their ability to "memorize" hard-to-classify samples.

3.2 Related Work

As we have already discussed in Section 2.2 from the previous chapter, there has recently been tremendous research interest into pruning techniques for neural networks. We follow the same categorization previously discussed in Chapter 2, where we consider progressive sparsification methods, applied post-training, and sparse regularization methods, which perform weight removal during the training process itself. We focus on sparse regularization methods, and in particular *sparse training*, although we will also compare against state-of-the-art progressive sparsification methods.

The general goal of *sparse training* methods is to perform both the forward (inference) pass *and the backpropagation* pass over a sparse support, leading to computational gains during

the training process. One of the first approaches to maintain sparsity throughout training is Deep Rewiring [BKML18], where SGD steps applied to positive weights are augmented with random walks in parameter space, followed by inactivating negative weights. To maintain sparsity throughout training, randomly chosen inactive connections are re-introduced in the "growth" phase. Sparse Evolutionary Training (SET) [MMS+18] introduces a non-uniform sparsity distribution across layers, which scales with the number of input and output channels, and trains sparse networks by pruning weights with smallest magnitude and re-introducing some weights randomly. Another sparse training method discussed in Section 2.2.2 from Chapter 2, is RigL [EGM+20], which can lead to state-of-the-art results, even compared to progressive sparsification methods applied to fully trained models; however, RigL requires periodically evaluating full gradients, and significant additional data passes (e.g. 5x) relative to the dense baseline, to reach competitive results. Top-KAST [JPR+20] alleviated the drawback of periodically having to evaluate dense gradients by updating the sparsity masks using gradients *of reduced sparsity* relative to the weight sparsity. The latter two methods set the state-of-the-art for sparse training: when executing for the same number of epochs as the dense baseline, they provide computational reductions the order of 2x, while the accuracy of the resulting sparse models is lower than that of leading post-training methods, executed at the same sparsity levels. To our knowledge, none of these methods have convergence guarantees.

Another approach towards faster training is *training sparse networks from scratch*. The masks are updated by continuously pruning and re-introducing weights. For example, [LSB+20] uses magnitude pruning after applying SGD on the dense network, whereas [DZ19] updates the masks by re-introducing weights with the highest gradient momentum. One of the top-performing methods in this category is STR [KRS+20], which we described in Section 2.2.2 from Chapter 2. One drawback of STR, however, is that the desired sparsity can not be explicitly imposed, and the network has low sparsity for a large portion of training. These methods can lead to only limited computational gains, since they either require dense gradients, or the sparsity level cannot be imposed. By comparison, our method provides models of similar or better accuracy at the same sparsity, with computational reductions. We also obtain dense models that match the baseline accuracy, with a fraction of the baseline FLOPs.

The idea of alternating sparse and dense training phases has been examined before in the context of neural networks, but with the goal of using temporary sparsification as a regularizer. Specifically, *Dense-Sparse-Dense (DSD)* [HPN+17] proposes to first *train a dense model to full accuracy*; this model is then sparsified via magnitude; next, optimization is performed over the sparse support, followed by an additional optimization phase over the full dense support. Thus, this process is used as a regularization mechanism for the dense model, which results in relatively small, but consistent accuracy improvements relative to the original dense model. In [JYFY16], the authors propose a similar approach to DSD, but alternate sparse phases during the regular training process. The resulting process is similar to AC/DC, but, according to their Algorithm 1, their procedure returns a dense model, which is later shown that it can be more accurate than the baseline. For this, the authors use relatively low sparsity levels, and gradually increase sparsity during optimization; they observe accuracy improvements, at the cost of increasing the total number of epochs of training. By contrast, our focus is on obtaining accurate *sparse* models, while reducing computational costs, and keeping the same number of training iterations as the dense baseline. Compared to [HPN+17] and [JYFY16], we also provide theoretical guarantees for the convergence of our method. Moreover, we show that AC/DC achieves good accuracy for highly sparse models, and on large-scale datasets and models. In addition, we provide results illustrating that our method generalizes to other sparsity patterns, e.g. the 2:4 semi-structured pattern [MLP+21]. Lastly, we show that with

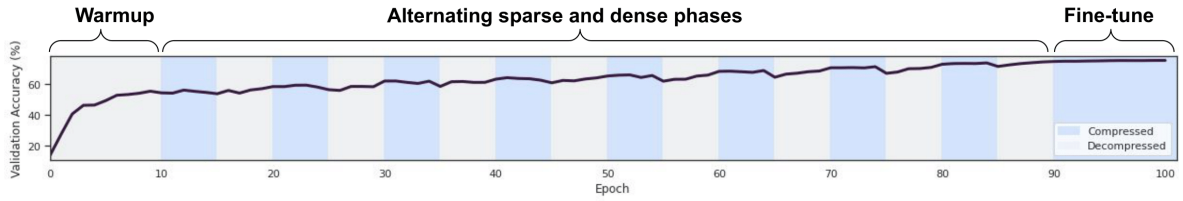


Figure 3.1: The AC/DC training process. After a short warmup we alternatively prune to maximum sparsity and restore the pruned weights. The plot shows the sparsity and validation accuracy throughout the process for a sample run on ResNet50/ImageNet at 90% sparsity.

AC/DC we can obtain *both* highly accurate dense and sparse models from a single training cycle, which enables us to analyze the differences at prediction level between these co-trained models.

Finally, parallel work by [MJS22] investigates a related approach, but focusing on low-rank decompositions for Transformer models. Both their analytical approach and their application domain are different to the ones of the current work.

3.3 Alternating Compressed / DeCompressed (AC/DC) Training

3.3.1 Background and Assumptions

Obtaining *sparse* solutions to optimization problems is an application of interest in several areas [CT06, BD08, Fou11], where the goal is to minimize a function $f : \mathbb{R}^N \to \mathbb{R}$ under sparsity constraints:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^N} f(\boldsymbol{\theta}) \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \leq k\,. \tag{3.1}$$

For the case of ℓ_2 regression, $f(\boldsymbol{\theta}) = \|b - A\boldsymbol{\theta}\|_2^2$, a solution has been provided by Blumensath and Davies [BD08], known as the *Iterative Hard Thresholding (IHT)* algorithm. Subsequent work [Fou11, Fou12, YLZ14] provided theoretical guarantees for the linear operators used in compressed sensing. The idea of IHT consists of alternating gradient descent (GD) steps and applications of a thresholding operator to ensure the ℓ_0 constraint is satisfied. More precisely, T_k is defined as the Top-K operator, which keeps the largest k entries of a vector $\boldsymbol{\theta}$ in absolute value, and replaces the rest with 0. The IHT update at step $t+1$ has the following form:

$$\boldsymbol{\theta}_{t+1} = T_k(\boldsymbol{\theta}_t - \eta \nabla f(\boldsymbol{\theta}_t)). \tag{3.2}$$

Notably, [YLZ14] propose introducing a "debiasing" phase after the Top-K operator is applied, which consists of training in the sparse support until convergence. However, most convergence results for IHT assume deterministic gradient descent steps. In the case of deep neural networks, which is also our setup, stochastic optimization methods are preferred, since they have been proven to lead to better solutions and are more practical for the large computational requirements of deep learning models. Therefore, this motivates us to analyze a stochastic version of IHT.

Stochastic IHT

We consider functions $f : \mathbb{R}^N \to \mathbb{R}$ for which we can compute stochastic gradients $g_{\boldsymbol{\theta}}$; these are unbiased estimators of the gradient $\nabla f(\boldsymbol{\theta})$. The *stochastic* IHT update can be defined as:

$$\boldsymbol{\theta}_{t+1} = T_k(\boldsymbol{\theta}_t - \eta g_{\boldsymbol{\theta}_t}). \tag{3.3}$$

As previously mentioned, one popular use of stochastic gradients is in the optimization of neural networks. In practice, the function f to be minimized is the average loss function over all training samples $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, defined as $f(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} f(\boldsymbol{\theta}; \boldsymbol{x}_i)$. In this case, the stochastic gradients $g_{\boldsymbol{\theta}}$ obtained via backpropagation take the form $\frac{1}{|B|}\sum_{i\in B} \nabla f(\boldsymbol{\theta}; \boldsymbol{x}_i)$, where $B \subset S$ is a sampled mini-batch. Our theoretical analysis covers this formulation, and we aim to prove strong convergence bounds for stochastic IHT, under common assumptions that arise in the context of training neural networks.

Analytical Assumptions for Stochastic IHT

In our analysis regarding the convergence of stochastic IHT, we use the following assumptions on the loss function f:

1. *Unbiased stochastic gradients with variance* $\sigma > 0$:

$$\mathbb{E}[g_{\boldsymbol{\theta}}|\boldsymbol{\theta}] = \nabla f(\boldsymbol{\theta}), \text{ and } \mathbb{E}[\|g_{\boldsymbol{\theta}} - \nabla f(\boldsymbol{\theta})\|^2] \leq \sigma^2$$

2. *Existence of a* k^**-sparse minimizer* $\boldsymbol{\theta}^*$:

$$\exists \boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \text{ s.t. } \|\boldsymbol{\theta}^*\|_0 \leq k^*$$

3. (t, β)*-smoothness*:

   For $\beta > 0$, the loss function f is β-smooth when restricted to t coordinates:

$$f(\boldsymbol{\theta} + \delta) \leq f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^\top \delta + \frac{\beta}{2}\|\delta\|^2, \text{ for all } \boldsymbol{\theta}, \delta \text{ s.t. } \|\delta\|_0 \leq t\,.$$

4. r*-concentrated Polyak-Łojasiewicz (*(r, α)*-CPL) condition*:

   For $\alpha > 0$ and number of indices r we assume the following holds:

$$\|T_r(\nabla f(\boldsymbol{\theta}))\|^2 \geq \frac{\alpha}{2}(f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*))\,, \text{ for all } \boldsymbol{\theta}. \tag{3.4}$$

The first assumption is standard in stochastic optimization, while the existence of sparse minimizers is a known property in over-parametrized deep learning models, and is the very premise of our study. In fact, many works in the area of sparse neural networks have shown that this assumption holds in practice, for example the Lottery Ticket Hypothesis [FC19, FDRC19], which we discussed in Section 2.2.2. Smoothness is also a standard assumption, which is used for example in [LSB+20]—we only require it along *sparse* directions, which is a strictly weaker assumption. The more interesting requirement for our convergence proof is the (r, α)-CPL condition in Equation (3.4), which we now discuss in more detail.

The standard Polyak-Łojasiewicz (PL) condition [KNS16] is common in non-convex optimization, and versions of it are essential in the analysis of deep neural networks training [LZB20, AZLS19]. Its standard form states that small gradient norm, i.e. approximate stationarity, implies closeness to optimum in function value. We require a slightly stronger version, in terms of the norm of the gradient contributed by its largest coordinates in absolute value. This restriction appears necessary for the success of IHT methods, as the sparsity enforced by the truncation step automatically reduces the progress ensured by a gradient step to an amount proportional to the norm of the Top-K gradient entries. This strengthening of the PL condition is supported both theoretically, by the mean-field view, which argues that gradients are sub-Gaussian [SM20], and by empirical validations of this behaviour [AHJ+18, SCCS19].

We now state our main analytical result regarding the convergence of stochastic IHT.

Theorem 3.3.1. *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function satisfying previous assumptions (1)-(4), with a k^*-sparse minimizer $\boldsymbol{\theta}^*$. Let $\beta > \alpha > 0$ be parameters, let $k = C \cdot k^* \cdot (\beta/\alpha)^2$ for some appropriately chosen constant C, and suppose that f is $(2k+3k^*, \beta)$-smooth and (k^*, α)-CPL.*

For initial parameters $\boldsymbol{\theta}_0$ and precision $\epsilon > 0$, given access to stochastic gradients with variance σ, stochastic IHT (3.3) converges in $O\left(\frac{\beta}{\alpha} \cdot \ln \frac{f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*)}{\epsilon}\right)$ iterations to a point $\boldsymbol{\theta}$ with $\|\boldsymbol{\theta}\|_0 \leq k$, such that

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}\right) - f\left(\boldsymbol{\theta}^*\right)\right] \leq \epsilon + \frac{16\sigma^2}{\alpha}.$$

Assuming a fixed objective function f and tolerance ϵ, we can obtain lower loss and faster running time by either increasing the support k demanded from our approximate minimizer $\boldsymbol{\theta}$ relative to the optimal k^*, or by reducing the gradient variance, for example by increasing the batch size. We provide a complete proof of this result in Appendix A.1.2. We note that the analysis approach also works in the absence of the CPL condition, in which case one can prove a version of the algorithm can find sparse nearly-stationary points. As a bonus, the existing analyses for IHT can be further simplified and adapted to the convex case. Both of these results can be found explained in detail in the Appendix of [PIVA21].

We note that Theorem 3.3.1 assumes the existence of a sparse minimizer $\boldsymbol{\theta}^*$, which, although supported empirically [FC19, FDRC19], could be a restrictive assumption. To address this, we can instead assume that the sparse point $\boldsymbol{\theta}^*$ is δ-close in function value to a minimizer $\boldsymbol{\theta}^{**}$, i.e. $f(\boldsymbol{\theta}^*) - f(\boldsymbol{\theta}^{**}) \leq \delta$, for some $\delta > 0$. Then, it is possible to still show convergence to $\boldsymbol{\theta}^{**}$ under the same number of iterations, such that $\mathbb{E}[f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^{**})] \leq \epsilon + O(\frac{\sigma^2}{\alpha}) + O(\delta)$.

Another interpretation of the result from Theorem 3.3.1 is in showing that, under our assumptions, *error feedback* [LSB+20] is not necessary for recovering good sparse minimizers; this has practical implications, as it allows us to perform fully-sparse back-propagation in sparse optimization phases. Next, we discuss our practical implementation, and its connection to these theoretical results.

3.3.2 AC/DC: Applying IHT to Deep Neural Networks

In this section we describe how the IHT algorithm can be adapted to training deep neural networks.

Algorithm 1 Alternating Compressed/Decompressed (AC/DC) Training

Require: Initial weights $\boldsymbol{\theta} \in \mathbb{R}^N$, data D, sparsity k, compression phases $\mathcal{C}$, decompression phases $\mathcal{D}$

Train the weights $\boldsymbol{\theta}$ for Δ_w epochs ▷ Warm-up phase
while epoch $\leq$ max epochs **do**
  if entered a compression phase **then**
    $\boldsymbol{\theta} \leftarrow T_k(\boldsymbol{\theta}, k)$ ▷ apply compression (Top-K) operator on weights
    $m \leftarrow \mathbb{1}[\boldsymbol{\theta}_i \neq 0]$ ▷ create masks
  end if
  if entered a decompression phase **then**
    $m \leftarrow \mathbb{1}_N$ ▷ reset all masks
  end if
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \odot m$ ▷ apply the masks (ensure sparsity for compression phases)
  $\tilde{\boldsymbol{\theta}} \leftarrow \{\boldsymbol{\theta}_i | m_i \neq 0, 1 \leq i \leq N\}$ ▷ get the support for the gradients
  for x mini-batch in D **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\tilde{\boldsymbol{\theta}}} f(\boldsymbol{\theta}; x)$ ▷ optimize the active weights
  end for
  epoch $\leftarrow$ epoch $+1$
end while
return $\boldsymbol{\theta}$

Implementation Details for AC/DC

From a practical perspective, AC/DC uses the regular training flow of a neural network, and common optimizers such as SGD with momentum [Qia99] or Adam [KB15], while preserving the standard training hyper-parameters. The main difference compared to the baseline training is that AC/DC will only periodically modify the *support* for optimization, depending on the compressed or decompressed phases. We present the pseudocode for AC/DC in Algorithm 1.

As specified in the algorithm, we partition the set of training epochs into *compressed* epochs $\mathcal{C}$, and *decompressed* epochs $\mathcal{D}$. We begin with a *dense warm-up* period of Δ_w consecutive epochs, during which regular dense (decompressed) training is performed. Subsequently, we start alternating *compressed optimization* phases of length Δ_c epochs each, with *decompressed (regular) optimization* phases of length Δ_d epochs each. The process completes on a compressed fine-tuning phase, returning an accurate sparse model. Alternatively, if our goal is to return a dense model matching the baseline accuracy, we take the last dense checkpoint obtained during alternation, and fine-tune it over the entire support for the remaining number of epochs, using the same training hyper-parameters used for the final compressed phase. In practice, we noticed that in some cases having a longer final decompressed phase of length $\Delta_D > \Delta_d$ improves the performance of the dense model, by allowing it to better recover the baseline accuracy after fine-tuning. A general illustration of the AC/DC training schedule can be seen in Figure 3.1. Moreover, the specific schedule used for most of our ImageNet experiments from Section 3.4, together with the evolution of the validation accuracy, can be visualized in Figure 3.2a.

In our experiments, we focus on the case where the compression operation is unstructured or semi-structured pruning. In this case, at the beginning of each sparse optimization (i.e. compressed) phase, we apply the Top-K operator globally across all of the prunable network weights to obtain a mask M over the weights $\boldsymbol{\theta}$. Following previous work [ZG17, GEH19,

EGM+20, KRS+20, SA20], all the parameters of the model, except the biases and the parameters of the normalization layers (e.g. Batch Normalization), are considered prunable. The mask M resulted after applying the Top-K operator will represent the sparse support over which optimization will be performed for the rest of the current sparse phase. At the end of the sparse phase, the mask M is reset to all-1s, so that the subsequent dense phase will optimize over the full dense support. Furthermore, once all weights are re-introduced, we found that it is beneficial to reset to 0 the gradient momentum term of the optimizer; this is particularly useful for the weights that were previously pruned, which would otherwise have stale versions of gradients.

Theoretical Guarantees for AC/DC

Comparing the differences between the stochastic IHT algorithm presented in Section 3.3.1 and the practical implementation described earlier, we note that several adjustments were needed to have a robust implementation of IHT for deep neural networks.

First, we note that each *decompressed phase* directly corresponds to a *deterministic/stochastic IHT* step, where, instead of a single gradient step in between consecutive truncations of the support, we perform several stochastic steps in the dense support. These additional steps improve the accuracy of the method in practice, and we can bound their influence in theory as well, although they do not necessarily provide better bounds. One explanation for the improvements obtained from using multiple dense stochastic gradient descent steps would be that these allow for the exploration of better masks.

This leaves open the interpretation of the *compressed phases*: for this, we note that the proof for Theorem 3.3.1 relies in showing that a single IHT step substantially decreases the expected value of the objective; using a similar argument, we can prove that more optimization steps over the sparse support can only improve convergence. Additionally, we show that our intuitions are also matched by theory. Namely, we present in Corollary 3.3.1 a convergence result for a variant of IHT closely following the practical implementation of AC/DC.

Corollary 3.3.1 (Convergence of AC/DC)**.** *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function that decomposes as $f(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\boldsymbol{\theta})$, and has a k^*-sparse minimizer $\boldsymbol{\theta}^*$. Let $\beta > \alpha > 0$ be parameters, let $k = C \cdot k^* \cdot (\beta/\alpha)^2$ for some appropriately chosen constant C, suppose that each f_i is (N, β)-smooth, and L-Lipschitz, and that f is (k^*, α)-CPL.*

Let Δ_c and B be integers, and let $\{D_1, \ldots, D_B\}$ be a partition of $[m]$ into B subsets of cardinality $O(m/B)$ each. Given $\boldsymbol{\theta}$, let $g_{\boldsymbol{\theta}}^{(i)} = \frac{1}{|D_i|}\sum_{j \in D_i} \nabla f_j(\boldsymbol{\theta})$.

Suppose we replace the IHT iteration with a dense/sparse phase consisting of

1. *Δ_c dense phases during each of which we perform a full pass over the data and update the parameters through the iteration $\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}^{(i)}$ for all $i \in [B]$, with an appropriate step size η;*

2. *a pruning step which applies the Top-K operator T_k over the weights θ;*

3. *an optional sparse training phase which fully optimizes f over the sparse support given by the Top-K operator*

For initial parameters $\boldsymbol{\theta}_0$ and precision $\epsilon > 0$, this algorithm converges in $O\left(\frac{\beta}{\alpha} \cdot \ln \frac{f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*)}{\epsilon}\right)$ dense/sparse phases to a point $\boldsymbol{\theta}$ with $\|\boldsymbol{\theta}\|_0 \leq k$, such that

$$f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*) \leq \epsilon + O\left(\frac{L^2}{\alpha}\right).$$

We note that the bounds presented in Corollary 3.3.1 are the same as for Theorem 3.3.1. Moreover, we observed empirically that although dense phases are necessary, training in the dense support for too long, before switching to the next compression phase, does not generally benefit the sparse model. Intuitively, this could suggest that it might be difficult to improve over the bounds from Corollary 3.3.1 and Theorem 3.3.1. However, the additional result in Corollary 3.3.1 confirms that the good experimental results obtained with AC/DC are theoretically motivated.

3.4 Experiments on Large-Scale Image Classification

3.4.1 General Setup

We performed experiments on the ImageNet dataset [RDS+15], where we trained ResNet50 [HZRS16] and MobileNetV1 [HZC+17] architectures using AC/DC; both these models are popular choices for model pruning benchmarks. Moreover, these architectures are also practically useful; for example, ResNet50 is often used for feature extraction in transfer learning [KSL19], as we will also discuss in Chapter 4, while the compact size of MobileNetV1 makes it a good architecture choice for testing the performance limits of pruning methods. Our goal is to examine the *validation accuracy* of the resulting sparse and dense models, versus the induced sparsity, as well as the number of FLOPs used for training and inference, relative to sparse training methods [EGM+20, JPR+20]. Additionally, we compare to state-of-the-art progressive pruning methods [SA20]. We also show that AC/DC enables training both accurate sparse and dense models with a single training cycle, at the expense of a small additional finetuning period, and we further examine prediction differences between the co-trained sparse–dense model pairs.

Unless otherwise specified, in all reported results, the models were trained for a fixed number of 100 epochs, using SGD with momentum. We use a cosine learning rate scheduler and training hyper-parameters following [KRS+20], but without label smoothing. For most of our experiments, we prune weights using global magnitude, in a single step; however, we also experiment with different pruning patterns, such as uniform, or semi-structured (e.g. 2:4 sparsity [MLP+21]). Similar to previous work, we did not prune biases, nor the Batch Normalization parameters. The sparsity level is computed with respect to all the parameters, except the biases and Batch Normalization parameters and this is consistent with previous work [EGM+20, SA20].

For all results, the AC/DC training schedule starts with a "warm-up" phase of dense training for 10 epochs, after which we alternate between compression and de-compression every 5 epochs, until the last dense and sparse phase. It is beneficial to allow these last two "fine-tuning" phases to run longer: the last decompression phase runs for 10 epochs, whereas the final 15 epochs are the compression fine-tuning phase. We reset SGD momentum at the beginning of every decompression phase. In total, we have an equal number of epochs of dense and sparse training, and we provide an illustration of the pruning schedule in Figure 3.2a. We use exactly

the same setup for both ResNet50 and MobileNetV1 models, which resulted in high-quality sparse models. To recover a dense model with baseline accuracy using AC/DC, we finetune the best dense checkpoint obtained during training; practically, this replaces the last *sparse* fine-tuning phase with a phase where the *dense* model is fine-tuned instead.

To reduce computation time, we train and evaluate most of the models using mixed precision (FP16) [MNA+18]. As a caveat, we mention that based on how mixed precision is implemented, a very small fraction of training steps (usually two or three per epoch) are skipped, due to possible infinite gradients after mixed precision re-scaling. In practice, this would have a negligible effect on our total training FLOPs calculation; however, we reproduced a subset of the experiments using full precision, and noticed small differences in validation accuracy of up to 0.2-0.3% between AC/DC trained with full or mixed precision.

3.4.2 Results on ResNet50

Comparing AC/DC with other pruning methods

We show that on the large scale ImageNet dataset, training the ResNet50 architecture with AC/DC results in very accurate sparse models, at a fraction of the training FLOPs of the dense baseline. Our results showing the Top-1 ImageNet validation accuracy, as well as inference and training FLOPs, in comparison with state-of-the-art sparse training or progressive sparsification methods are presented in Table 3.1 for medium sparsity (80% and 90%) and Table 3.2 for high sparsity (95% and 98%). Overall, AC/DC achieves higher validation accuracy than any of the state-of-the-art sparse training methods (RigL and Top-KAST), when using the same number of epochs; notably, we will also consider later on the versions with extended training time for RigL and Top-KAST (e.g. we refer to RigL 5x as the version with 5 times more training iterations, compared to the baseline). At the same time, due to dense training phases, AC/DC has higher FLOP requirements relative to RigL or Top-KAST at the same sparsity.

At medium sparsities (80% and 90%), AC/DC sparse models are slightly less accurate than the state-of-the-art post-training methods (e.g. WoodFisher), by small margins. The situation is reversed at higher sparsities, where AC/DC produces more accurate models: the gap to the second-best methods (WoodFisher / Top-KAST) is of more than 1% at 95% and 98% sparsity. We note that the AC/DC results we report in Tables 3.1 and 3.2 are obtained using global magnitude pruning, while the other methods use different pruning distributions; for example, RigL and Top-KAST use uniform magnitude pruning, while keeping the first or last layers dense; we also compare against the version or RigL with the Erdős-Rényi-Kernel (ERK) sparsity distribution. However, we will show in what follows that AC/DC achieves good results also with other sparsity distributions, such as uniform or semi-structured.

Of the existing sparse training methods, Top-KAST is closest in terms of validation accuracy to our sparse model, at 90% sparsity. Since Top-KAST uses uniform magnitude pruning and keeps the first and last layers dense, we further compare with AC/DC using the same sparsity pattern. For 90% uniform sparsity, we obtain 75.04% accuracy with AC/DC, which is still slightly above Top-KAST (74.76% accuracy). Moreover, we obtain 73.28% accuracy for 95% uniform sparsity, which is well above all the other pruning methods used for comparison (e.g. Top-KAST achieves 71.96% accuracy). Overall, we observed that keeping the first and last layers dense has an important positive impact on the model accuracy, at only a very small additional computational cost (e.g. 3% of the total FLOPs). For example, when using global magnitude pruning, with first and last layers dense, the 90% sparse AC/DC improved to

75.64% accuracy, while for 95% and 98% target sparsities the accuracy improved to 74.16% and 71.27%, respectively.

We further note that the results for both Top-KAST and RigL improve substantially with extended training. For example, Top-KAST obtains better results at 98% sparsity (≈69% accuracy) when increasing the number of training epochs two times, at considerably fewer training FLOPs, compared to AC/DC (e.g. 15% of the dense FLOPs vs. 46% for AC/DC). We show a more detailed comparison between AC/DC and Top-KAST with extended training time in Appendix Table A.5. For fairness, we compared in this setting against all methods on a fixed number of 100 training epochs. However, we show in Section 3.4.4 that extending the training time in AC/DC substantially improves the results.

Method	Sparsity (%)	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	0	76.84	8.2	3.14
80% Target Sparsity				
AC/DC	80	76.3 ± 0.1	0.29 ×	0.65 ×
RigL	80	74.6 ± 0.06	0.23 ×	0.23 ×
RigL ERK	80	75.1 ± 0.05	0.42 ×	0.42 ×
Top-KAST	80 fwd, 50 bwd	75.03	0.23 ×	0.32 ×
STR	79.55	76.19	0.19 ×	-
WoodFisher	80	**76.76**	0.25 ×	-
90% Target Sparsity				
AC/DC	90	**75.03 ± 0.1**	0.18 ×	0.58 ×
RigL	90	72.0 ± 0.05	0.13 ×	0.13 ×
RigL ERK	90	73.0 ± 0.04	0.24 ×	0.25 ×
Top-KAST	90 fwd, 80 bwd	74.76	0.13 ×	0.16 ×
STR	90.23	74.31	0.08 ×	-
WoodFisher	90	**75.21**	0.15 ×	-

Table 3.1: (ResNet50/ImageNet) Results for *medium* (80% and 90%) sparsity.

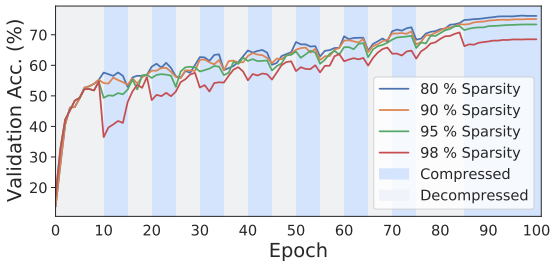
Method	Sparsity (%)	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	0	76.84	8.2	3.14
95% Target Sparsity				
AC/DC	95	**73.14 ± 0.2**	0.11 ×	0.53 ×
RigL	95	67.5 ± 0.1	0.08 ×	0.08 ×
RigL ERK	95	69.7 ± 0.17	0.12 ×	0.13 ×
Top-KAST	95 fwd, 50 bwd	71.96	0.08 ×	0.22 ×
STR	94.8	70.97	0.04 ×	-
WoodFisher	95	72.12	0.09 ×	-
98% Target Sparsity				
AC/DC	98	**68.44 ± 0.09**	0.06 ×	0.46 ×
Top-KAST	98 fwd, 90 bwd	67.06	0.05 ×	0.08 ×
STR	97.78	62.84	0.02 ×	-
WoodFisher	98	65.55	0.05 ×	-

Table 3.2: (ResNet50/ImageNet) Results for *high* (95% and 98%) sparsity.

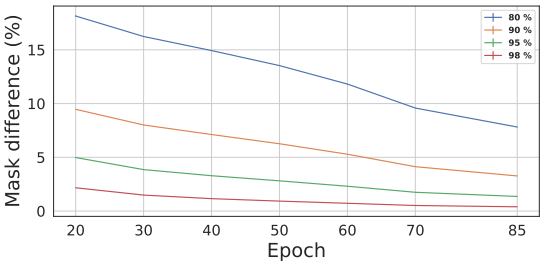
Training Dynamics and Evolution of Pruning Masks

We further analyze the evolution of both model accuracy and difference between consecutive pruning masks, during training. We observe that when measuring both the train and validation accuracy, after each pruning phase, the accuracy drops at the beginning of a decompression phase, when all weights are reintroduced; however, this drop recovers quickly and, more importantly, each decompressed training phase is beneficial for the next pruning stage. At 98% sparsity in particular, it is easiest to see that dense phases enable the exploration of better pruning masks, which ensure that the sparse model improves continuously during training. Moreover, when training with high sparsity levels (e.g. ≥95%), we observe a large drop in accuracy after pruning, which recovers quite quickly in the subsequent decompression phase. We believe these particularities are due to the high learning rate, which is maintained for a large period of the training cycle, and which in turn encourages more exploration and faster recovery from sub-optimal regions of the loss landscape; the evolution of the validation accuracy is presented in Figure 3.2a, and the train accuracy follows a similar pattern.

The mask dynamics, measured by the relative change between two consecutive compression masks, have an important influence on the AC/DC training process. Namely, more changes between consecutive compression masks typically imply more exploration of the weights' space, and faster recovery from sub-optimal pruning decisions, which in turn results in more accurate sparse models. As can be seen in Figure 3.2b, the relative mask difference between consecutive compression phases decreases during training, but it is important to be maintained at a non-trivial level. Interestingly, as training progresses, we noticed that AC/DC also induces



(a) Sparsity pattern and validation accuracy vs. number of epochs, at different sparsity levels (80%-98%).



(b) Differences between consecutive masks updates during AC/DC training at different target sparsities (80%-98%).

Figure 3.2: (ImageNet/ResNet50) Accuracy vs. sparsity during training (left) and the corresponding difference between consecutive masks updates (right).

structured sparsity, as more neurons and convolutional filters get pruned. We describe this phenomenon in more detail in Appendix A.2.1. Moreover, as we will see in Chapter 4, the inherent structured sparsity present in AC/DC models can also influence their generalization ability, for example when they are used for transfer learning on different data distributions.

Results on Semi-Structured Sparsity

In addition to unstructured sparsity obtained through global and uniform magnitude pruning, we further show that AC/DC can achieve good results with semi-structured sparsity patterns. Namely, we experiment with the 2:4 sparsity pattern [MLP+21], which we described in Chapter 2, Section 2.2.3, and which can result in practical inference speedups on the NVIDIA Ampere GPU architecture. We applied AC/DC to the 2:4 pattern, performing training from scratch and obtained sparse models with $76.64\% \pm 0.05$ validation accuracy, i.e. slightly below the baseline. Furthermore, the dense-finetuned model fully recovers the baseline performance (76.85% accuracy). We additionally experimented with using AC/DC with global pruning at 50%; in this case we obtained sparse models that slightly improve the baseline accuracy to 77.05%. This confirms our intuition that AC/DC can act as a regularizer at lower sparsity levels, which has been previously observed also in [HPN+17].

Practical Speedups

One remaining question concerns the potential of sparsity to provide real-world speedups. While this is an active research area, e.g. [EDGS20], we partially address this concern in Appendix Table A.3, by showing inference speedups for our models on a CPU inference platform supporting unstructured sparsity [Dee21]: for example, our 90% sparse ResNet50 model provides 1.75x speedup for real-time inference (batch-size 1) on a resource-constrained processor with 4 cores, and 2.75x speedup on 16 cores at batch size 64, versus the dense model.

3.4.3 Results on MobileNet

In addition to the results on ResNet50, we further show that training with AC/DC can result in accurate sparse models on other architectures. Namely, we prune the MobileNetV1 [HZC+17] model with AC/DC, using the same training recipe and under the same hyperparameter setting as for ResNet50. Therefore, the hyperparameter tuning also in this case was minimal, as we used the same hyper-parameters as for training the dense baseline in [KRS+20], except for

label smoothing, which we did not use. On a training budget of 100 epochs, our method finds sparse models with higher Top-1 validation accuracy than existing sparse training methods, on both 75% and 90% sparsity levels. We provide a comparison between AC/DC and existing pruning methods in Table 3.3; our comparison also includes the RigL models trained for extended number of epochs (e.g. 2x or 5x the number of passes through the training data).

Notably, the only sparse training method which obtains higher accuracy for the same sparsity level is the version of RigL [EGM+20] which trains for five times more epochs than the dense baseline. In this setup, however, RigL also uses more computation than the dense baseline model. We limit ourselves to a fixed number of 100 epochs, the same used to train the dense baseline, which would allow for savings in training time. Moreover, we note that RigL does not prune the first layer and the depth-wise convolutions, whereas for the results reported we do not impose any sparsity restrictions. Overall, we found that keeping these layers dense improved our results on 90% sparsity by more than 0.5%. Our results are quite close to RigL 2x, with half the training epochs, and less training FLOPs. We note that RigL 5x obtains higher validation accuracy, and for 75% sparsity it even matches the baseline; however, this variant of RigL also uses $2.6\times$ and $1.5\times$ the dense baseline training FLOPs for 75% and 90% sparsities, respectively.

Moreover, we note that AC/DC is surpassed by progressive sparsification methods, such as M-FAC [FKA21] (discussed in Section 2.2.2), which achieves state-of-the-art results on MobileNetV1 for both 75% and 90% sparsity, when training for 100 epochs. However, we note that M-FAC requires access to a pretrained dense model and, despite its efficiency over other second-order methods, it still has higher computational and memory requirements during training. We further mention that M-FAC is pruned to 89% sparsity, compared to AC/DC or RigL, which are pruned at 90%.

Method	Top-1 Acc. (%)	GFLOPs Inference	EFLOPs Train
Dense	71.78	1.1	0.44
75% Sparsity			
AC/DC	70.1 ± 0.14	0.34×	0.65×
AC/DC*	70.52	0.36×	0.66×
RigL ERK 1x	68.39	0.52×	0.53×
RigL ERK 2x	70.49	0.52×	1.05×
RigL ERK 5x	**71.9**	0.52×	2.63×
M-FAC	70.9	N/A	N/A
90% Sparsity			
AC/DC	66.1 ± 0.18	0.18×	0.56×
AC/DC*	66.84	0.21×	0.58×
RigL ERK 1x	63.58	0.27×	0.29×
RigL ERK 2x	65.92	0.27×	0.59×
RigL ERK 5x	**68.1**	0.27×	1.47×
M-FAC	67.2	N/A	N/A

Table 3.3: (MobileNetV1/ImageNet) Validation accuracy for sparse models, together with inference and train FLOPs, where provided. The ($\star$) indicates that the first layer and depth-wise convolutions were kept dense.

Method	Sparsity (%)	Perplexity Sparse	Perplexity Dense	Perplexity Finetuned Dense
Dense	0	-	18.95	-
80% Sparsity				
AC/DC	80	20.65	20.24	19.54
AC/DC	80, 50 embed.	20.83	20.25	19.68
Top-KAST	80, 0 bwd	**19.8**	-	-
Top-KAST	80, 60 bwd	21.3	-	-
90% Sparsity				
AC/DC	90	**22.32**	21.0	20.28
AC/DC	90, 50 embed.	**22.84**	21.34	20.41
Top-KAST	90, 80 bwd	25.1	-	-

Table 3.4: Transformer-XL/WikiText results for AC/DC and Top-KAST pruned for 80% and 90% sparsity targets. For AC/DC we additionally measure the quality of the resulting dense models before and after finetuning.

3.4.4 Improvements from Extended Training Time

In this section we show that we can obtain better sparse models when extending the training time with AC/DC. Substantial improvements from extended training have also been observed

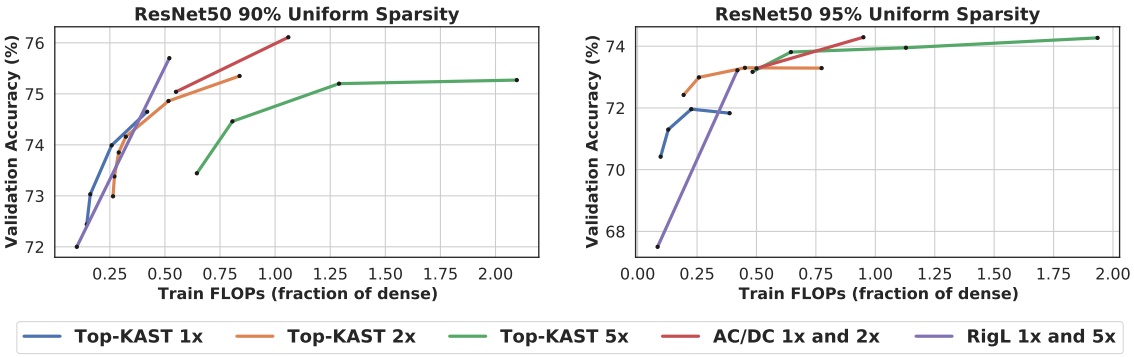


Figure 3.3: (ImageNet/ResNet50) Training FLOPs vs validation accuracy for AC/DC, RigL and Top-KAST, with uniform sparsity, at 90% and 95% sparsity levels. Inference FLOPs are the same for all methods.

with other pruning methods, such as RigL [EGM+20] or Top-KAST [JPR+20]. Our experiments with extended training for AC/DC are performed on the ImageNet dataset [RDS+15], using the ResNet50 architecture [HZRS16].

Results for Uniform Sparsity

First, we provide a direct comparison between AC/DC and Top-KAST and RigL, in terms of the validation accuracy achieved on ImageNet vs. the number of training FLOPs. We report results at uniform sparsity, since this ensures that the inference FLOPs will be the same for all methods considered. For AC/DC and Top-KAST, the first and last layers are kept dense, whereas for RigL, only the first layer is kept dense; however, this has a negligible impact on the number of FLOPs. We extend the number of training iterations for AC/DC at 90% and 95% sparsity two times; this is done by extending both the learning rate scheduler and training recipe, while preserving the duration of each compressed and decompressed phase, as well as other training hyperparameters (e.g. momentum or weight decay).

The comparison between AC/DC, Top-KAST and RigL presented in Figure 3.3 shows that AC/DC with the standard number of training epochs is similar to or surpasses Top-KAST 2x at 90% and 95% sparsity, as well as RigL 5x at 95% sparsity, both in terms of training FLOPs and validation accuracy. Moreover, we highlight that extending the number of training iterations two times results in AC/DC models with uniform sparsity that surpass all existing methods at both 90% and 95% sparsity; namely, we obtain 76.1% and 74.3% validation accuracy with 90% and 95% uniform sparsity, respectively. The only exception is Top-KAST 5x, which can achieve up to 74.3% accuracy for 95% sparsity.

Method	Sparsity 80%	90%	95%
AC/DC	76.3 ± 0.1	75.0 ± 0.1	73.1 ± 0.2
AC/DC 3x	77.5	76.8	75.3
AC/DC 5x	N/A	77.2	N/A
RigL ERK 1x	75.1 ± 0.05	73.0 ± 0.04	69.7 ± 0.17
RigL ERK 5x	77.1 ±0.06	76.4 ± 0.05	74.5 ± 0.09

Table 3.5: (ImageNet/ResNet50) Validation accuracy (%) of sparse models trained for extended number of iterations. AC/DC models were pruned using global magnitude pruning.

Method	Sparsity 90%	95%
AC/DC	74.7	72.8
AC/DC 2x	75.8	N/A

Table 3.6: (ImageNet/ResNet50) Validation accuracy (%) of AC/DC models when reducing the dense phases to two epochs each. Results are for uniform sparsity.

Results for Global Sparsity

Inspired by the encouraging results for AC/DC 2x, we examine whether extending the training time even longer can further improve the resulting sparse models. Namely, we extend the AC/DC training time three times, for 80%, 90% and 95% target sparsity levels; additionally, we train AC/DC for five times more epochs, at 90% target sparsity. For this setup, we use global magnitude pruning and compare against RigL [EGM+20] pruned using ERK sparsity distribution, which results in higher accuracy for RigL models, compared to uniform pruning.

Our results presented in Table 3.5 show that AC/DC 3x models substantially surpass RigL ERK 5x at all sparsity levels considered. Moreover, when training AC/DC 5x to match the training iterations in RigL ERK 5x, the results at 90% sparsity still continue to improve, with AC/DC 5x reaching 77.3% accuracy. Interestingly, training sparse models longer not only improves their accuracy on the validation set, but also improves their generalization when they are used for finetuning on different tasks from the ones used initially for training; we explore these properties in more detail in Section 4.4.5 from Chapter 4.

3.5 Additional Experimental Validations

3.5.1 Reducing the Dense Phases in AC/DC

We note that compared to purely sparse training methods, such as Top-KAST or RigL, AC/DC can have higher computational requirements, as it requires periods of dense training. However, the length of the dense phases can be decreased, with a relatively small impact on the accuracy of the sparse model. We show this on the ResNet50 architecture trained on ImageNet.

Specifically, we use dense phases of two instead of five epochs in length, and we no longer extend the final decompressed phase prior to the finetuning phase. For 90% global sparsity, this resulted in 74.6% validation accuracy for the sparse model, using 44% of the baseline FLOPs. Similarly, for uniform sparsity, we obtain 74.7% accuracy on the 90% sparse model, with 40% of the baseline FLOPs; this value can be further improved to 75.8% validation accuracy when extending two times the number of training iterations. Furthermore, at 95% uniform sparsity, we reach 72.8% accuracy with 35% of the baseline training FLOPs. We present these results in Table 3.6.

When further reducing the length of a decompression phase to a single epoch, we observe a more pronounced degradation of the sparse model; namely, at 90% target sparsity, we obtain models with 74.2% ImageNet validation accuracy, which improves to 75.5% when extending the training schedule two times; furthermore, when running AC/DC at 95% target sparsity for 200 epochs, using dense phases of one epoch each (except the warm-up phase of 10 epochs), we obtain 73.5% validation accuracy for the sparse model. These results motivate us to further investigate the limits of the dense training phases in AC/DC, and whether, for example, increasing the number of total training iterations, while keeping the dense phases very short, would enable us to still obtain improvements for the sparse models.

3.5.2 Language Modeling Results

In addition to image classification tasks, we show that AC/DC can also be used for language modeling. Specifically, we use AC/DC to train sparse models on the Transformer-XL [DYY+19] model on the WikiText-103 dataset [MXBS16]. More details on the Transformer-XL model

and the WikiText-103 dataset are provided in Section 2.1.3 from Chapter 2. We integrated the implementation provided by NVIDIA [NVI21], which follows closely the original implementation in [DYY+19]. For training, we use the standard model configuration with 18 layers and 285M parameters, trained using the Lamb optimizer [YLR+19] and standard hyper-parameters.

We choose the Transformer-XL model architecture trained on WikiText-103 as it was previously used with Top-KAST [JPR+20], which allows a direct comparison between AC/DC and Top-KAST, also on natural language processing tasks. Similar to Top-KAST, we did not prune the embedding layers, as this greatly affects the quality, without reducing computational cost. However, for completeness, we do provide results when embeddings are pruned to 50% sparsity. Our sparse training configuration consists in starting with a dense warm-up phase of 5 epochs, followed by alternating between compression and decompression phases every 3 epochs; we follow with a longer decompression phase between epochs 33-39, and end with a compression phase between epochs 40-48. Unlike the image classification setup, for these experiments we did not reset the statistics involving exponential moving average of gradients before entering a dense phase. Moreover, we show that we can improve the quality of the resulting dense model with additional finetuning, which involves replacing replacing the last compression phase with regular dense training.

The results for the comparison between AC/DC and Top-KAST at 80% and 90% target sparsities, as well as for the AC/DC dense models, are shown in Table 3.4. Relative to Top-KAST, our approach provides substantially improved test perplexity at 90% sparsity, as well as better results at 80% sparsity, when sparse backpropagation is used for Top-KAST. These results on language modeling tasks confirm that AC/DC is scalable and extensible; furthermore, we note that our hyper-parameter tuning for this experiment was minimal.

3.6 Generalization properties of AC/DC

As presented in the previous sections, AC/DC trains accurate models at different sparsity levels, on tasks ranging from image classification to language modeling. Additionally, we show that the AC/DC training dynamics reveal some interesting properties related to the generalization of sparse neural networks. For example, AC/DC enables co-training sparse-dense model pairs which allow for a better study of the differences in predictions between sparse and dense models; we explore this property in detail in Section 3.6.1. Additionally, it has been hypothesized that sparse models are less prone to overfitting and can lead to better generalization, compared to their dense counterparts. Taking advantage of the sparse-dense co-training through AC/DC, we examine the differences in generalization between sparse and dense models, through the lens of random labels memorization in Section 3.6.2 and find that, indeed, sparse models are less likely to memorize random data.

3.6.1 Analyzing Differences Between Sparse and Dense AC/DC Models

Results for the Dense Models

One advantage of AC/DC is that it provides *both* sparse and dense models at cost *below* that of a single dense training run. For example, for medium sparsity, the accuracy of the dense-finetuned model is very close to the dense baseline. To obtain highly accurate dense models from an AC/DC training cycle, we fine-tune the resulting dense AC/DC model before

the final pruning phase, for a small number of epochs. Namely, we start from the best dense baseline, which is usually obtained after 85 training epochs, and replace the final compression phase with 15 epochs of regular dense training; we use the same learning rate scheduler and keep all other training hyper-parameters the same. Using this recipe on ResNet50 models trained on ImageNet, we find that when training with 80% target sparsity we recover the dense baseline accuracy, while for 90% target sparsity we are slightly below the baseline. We note that for 90% sparsity, when the first and last layers are kept dense, our fine-tuned dense model recovers the baseline accuracy fully. The results for the dense models, before and after finetuning, together with the baseline accuracy, are presented in Table 3.7; here, ($\star$) denotes that the first and last layers of the network are kept dense, in the case of ResNet50, or the first layer and depth-wise convolutions are dense, for MobileNetV1.

Target Sparsity	Accuracy Dense (%)	Accuracy Finetuned (%)
0%	76.84	-
80%	73.82 ± 0.02	76.83 ± 0.07
90%	73.25 ± 0.16	76.56 ± 0.1
90%*	73.66	76.85

Table 3.7: (ImageNet/ResNet50) Validation accuracy of AC/DC dense models, before and after the final dense finetuning phase.

Sparsity	Accuracy Dense (%)	Accuracy Finetuned (%)
0%	71.78	-
75%	68.37 ± 0.18	71.41 ± 0.15
90%	67.34 ± 0.14	70.76 ± 0.19
90%*	67.69	70.91

Table 3.8: (ImageNet/MobileNetV1) Validation accuracy of AC/DC dense models, before and after the final dense finetuning phase.

Additionally, we observe that the resulting AC/DC dense models, before finetuning, have a small percentage of zero-valued weights. We believe this is most likely caused by "dead" neurons or convolutional filters resulted after each compression phase; the corresponding weights do not get re-activated during the dense stages, as they can no longer receive gradients. We provide more details on this phenomenon in Appendix Table A.1.

Similar to ResNet50, dense models obtained through AC/DC training on MobileNetV1 are able to recover the baseline accuracy after additional finetuning. We performed finetuning identically to the ResNet50 experiments and observe that the dense AC/DC models resulted from training for 75% target sparsity almost recovered the baseline accuracy while for 90% target sparsity, the gap between dense AC/DC models and the baseline is around 1%. The results for the (fine-tuned) AC/DC dense models, together with the baseline, are presented in Table 3.8; as previously mentioned, ($\star$) indicates that the first layer and depth-wise convolutions were never pruned.

Method	Sparse Top-1 Accuracy (%)	Dense Top-1 Accuracy (%)	Sparse-Dense Agreement (%)	Sparse-Dense Cross-entropy
80% Sparsity				
AC/DC	76.3 ± 0.1	76.8 ± 0.07	89.8 ± 0.3	0.85 ± 0.005
SparseVD	75.3	75.2	98.6	-
GMP	76.4	76.9	86.0	1.03
90% Sparsity				
AC/DC	75.0 ± 0.1	76.6 ± 0.09	86.8 ± 1.5	1.02 ± 0.004
SparseVD	73.8	73.6	98.3	-
GMP	74.7	76.9	83.5	1.29

Table 3.9: (ImageNet/ResNet50) Sample agreement between ResNet50 sparse and dense models

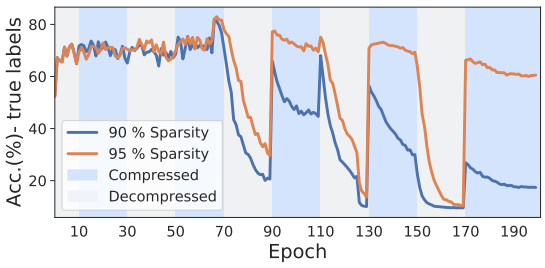


Figure 3.4: (ResNet20/CIFAR-10) Percentage of samples with corrupted training labels classified to their *true* class.

Sparse-Dense Output Comparison

The accurate sparse-dense model pairs obtained from a single AC/DC training cycle offer a good opportunity to study the differences in the predictions between these two models.

For this purpose, we employ two different metrics; first, we examine *sample-level agreement*, which simply measures on how many of the validation samples the predictions between the sparse and dense AC/DC models agree. Next, we also measure the differences in the per-sample probability distributions, by computing the *cross-entropy* between the softmax probability distributions of sparse and dense AC/DC models. To understand the relative sizes of these differences, we compare against two baselines. First, we compared the similarity of a fully trained dense model with the sparse models obtained from gradual magnitude pruning (GMP) trained over 100 epochs; in this case, the dense model was used as initialization for the GMP training cycle. Second, we compare with the sparse-dense model couples obtained by Sparse Variational Dropout (SparseVD) [MAV17]; in SparseVD, the dense model is trained to optimize a variational lower bound, and at the end of training, a large proportion of the weights can be pruned in a single step, without affecting the accuracy of the dense model. However, SparseVD incurs a computational overhead during training, as it uses two times more parameters.

The results in Table 3.9 show the ImageNet validation accuracies for the corresponding sparse and dense model pairs for AC/DC, SparseVD and GMP, as well as the sample agreement and prediction cross-entropy; in the case of AC/DC, we use the finetuned dense models. We note that AC/DC and GMP show comparable accuracy for both their sparse and dense models, while the resulting SparseVD models have lower accuracy, at both 80% and 90% sparsity. Moreover, the sparse and dense SparseVD models agree in over 98% of their predictions as measured on the ImageNet validation set, and the dense model has the same quality as the sparse model. In comparison, AC/DC with finetuning produces dense models of validation accuracy that is comparable to that of a dense model trained without any compression, and that therefore do differ from their sparse co-trained counterparts. When examining the sample-agreement, AC/DC co-trained model pairs consistently agree on more samples relative to GMP: for example, on the 80%-pruned ResNet50 model, the AC/DC model pair agrees on the Top-1 classification of 90% of validation samples, whereas the GMP models agree on 86% of the samples. The differences are better seen in terms of validation error (10% versus 14%), which indicate that the dense baseline and GMP model disagree on 40% more samples compared to the AC/DC models. A similar trend holds for the *cross-entropy* between model outputs, which is about 20% lower for AC/DC models, compared to GMP. This is a potentially useful side-effect of the method; for example, in constrained environments where sparse models are needed, it is important to estimate their similarity to the dense ones.

3.6.2 Memorization Experiments

Lastly, we analyze the differences between sparse and dense AC/DC models based on their ability to memorize samples with random labels. As discussed in Section 2.1.4 and Section 2.2.4, neural networks have the capacity to memorize the training data [ZBH+17], including random labels, which seems to be at odds with their good generalization abilities. Moreover, several works have hypothesized that sparse models might have improved generalization abilities over the dense ones [JCR+22, AGNZ18, HPN+17]. We take advantage of the existence of sparse-dense model pairs in AC/DC, to directly study the evolution of generalization for both of these models, during the training cycle.

For this purpose, we apply AC/DC to ResNet20 trained on a variant of CIFAR-10 where

a subset of 1000 samples have randomly corrupted class labels, and examine the accuracy on these samples during training. We consider 90% and 95% sparsity AC/DC runs. For this experiment, all models were trained without using data augmentation. We present our findings in Figure 3.4, where we measure the accuracy on the perturbed subset of the training set, with respect to the *true, un-corrupted* label. During early training and during *sparse phases*, the network tends to classify corrupted samples to their *true class*, "ignoring" label corruption. However, during dense phases, as training progresses and learning rate decreases, dense networks tend to "memorize" these samples, assigning them to their corrupted class. This phenomenon is even more prevalent at 95% sparsity, where the sparse network is less capable of memorization. In Appendix A.2.2 we discuss these findings in more detail, and we additionally show a similar analysis when data augmentation is used during training.

3.7 Conclusion, Limitations, and Future Work

We introduced AC/DC—a method for co-training sparse and dense models, with theoretical guarantees. Experimental results show that AC/DC improves upon the accuracy of previous sparse training methods, and obtains state-of-the-art results at high sparsities. Importantly, we recover near-baseline performance for dense models and do not require extensive hyper-parameter tuning. We also show that AC/DC has potential for real-world speed-ups in inference and training, with the appropriate software and hardware support. When training with moderate sparsity, the method has the advantage of returning both an accurate dense model, and a sparse one. Our model output analysis confirms the intuition that sparse training phases act as a regularizer, preventing the (dense) model from memorizing corrupted samples. At the same time, they prevent the memorization of *hard samples*, which can affect accuracy.

The main limitations of AC/DC are its reliance on dense training phases, which restricts the achievable training speedup, and the need for tuning the length and frequency of sparse/dense phases. We believe the latter issue can be addressed with more experimentation (we show some preliminary results in Section 3.5.1); however, both the theoretical results and the output analysis suggest that dense phases may be *necessary* for good accuracy. We plan to further investigate this in future work, together with applying AC/DC to other compression methods, such as quantization, as well as leveraging sparse training on hardware that could efficiently support it, such as Graphcore IPUs [Gra21].

CHAPTER 4

Study on the Transferability of Sparse Convolutional Neural Networks

As previously discussed in Chapter 2, model pruning has received a good amount of academic and industrial interest in recent years. Moreover, as we have also shown in Chapter 3, it is possible to train accurate sparse neural networks at a lower cost compared to dense models.

The efforts of obtaining smaller-footprint models and reducing the training costs associated with them are focused on solving an important challenge in deep learning, which is enabling the deployment of large models on edge devices, such as, for example, mobile phones or laptops. As these devices may naturally encounter different data distributions, it is tempting to ask how sparse models would perform when they are used for *transfer learning*, broadly defined as leveraging information from some baseline "upstream" ("pretrained") task in order to perform better on a "downstream" ("finetuning") task. In this chapter, we want to provide a justified answer to this question, by mainly focusing on a prototypical transfer learning setup, used in many previous works, such as [KSL19, SIE+20]. Namely, starting from models trained and pruned on the ImageNet dataset [RDS+15] with 1000 classes, we finetune them on several different target tasks. We believe such a study focusing on sparse models is of great practical interest, as there are many powerful baselines developed for sparse models, as well as increasingly available hardware support to leverage speed-up from sparsity.

4.1 Motivation and Outline

The motivation for our study on the transferability of sparse models is both practical—sparse transfer can provide speedups for both inference and training on the downstream model—and analytical, as we aim to shed light on the impact of sparsity on the resulting features and in general, how well can sparse models adapt to new data distributions.

Our study will consider the two common transfer learning variants we described in Chapter 2, Section 2.1.5, namely *full finetuning*, where all unpruned weights can be optimized during transfer, and *linear finetuning*, where only the final linear layer of the model is finetuned downstream. While both are popular, we will see that they can lead to different results. We additionally explore *inference-time* speedups using a sparsity-aware inference engine [Dee21, KKG+20], and for the first time examine *training-time speed-up* achievable for linear finetuning

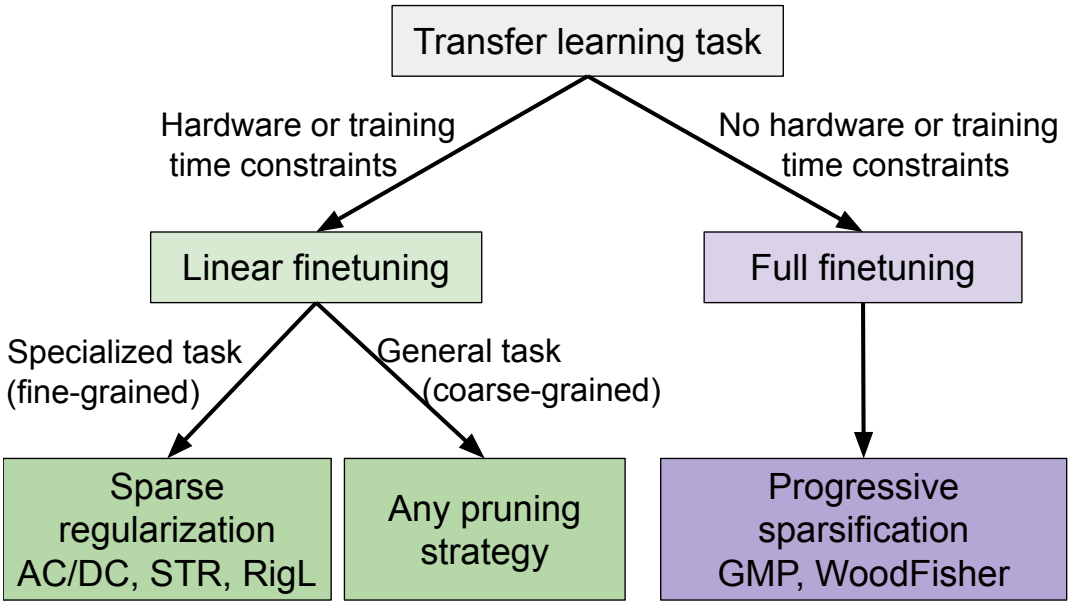


Figure 4.1: Overview of a suggested decision process when selecting the finetuning and pruning methods to maximize performance and accuracy when doing transfer learning on pruned models.

via sparse models. Furthermore, we analyze the impact of different pruning methods and task characteristics on transfer performance.

We consider the top-performing unstructured pruning methods in terms of ImageNet accuracy, roughly split into three categories, which we also discussed in more detail in Chapter 2, Section 2.2.2. Namely, the first category is given by *progressive sparsification* methods, which start from an accurate *dense* baseline and proceed to gradually remove weights, followed by finetuning; for the second category we consider *sparse regularized training* methods, which perform network compression, and possibly network re-growth, during the training process itself, and, finally, for the third category, we consider Lottery Ticket Hypothesis (LTH)-style methods [FC19, FDRC19, CFC$^+$20, CFC$^+$21], which emphasize the *discovery* of sparse sub-networks, to obtain good accuracy when re-trained from scratch. We emphasize that this categorization is approximate, as, for example, LTH methods could be viewed as a special case of progressive sparsification, where a specific finetuning approach is applied. Moreover, it is not uncommon to combine approaches, such as regularization and progressive sparsification [HABN$^+$21]. We provide a comparison of the efficacy of these different approaches in the context of transfer learning, by considering multiple methods from each category. To our knowledge, this is the first such detailed study.

Our main target application is given by twelve classic transfer datasets, described in Table 4.2, ranging from general datasets, to more specialized ones. We mainly focus on the classic ResNet50 [HZRS16] model, but we extend our analysis to ResNet18, ResNet34 and MobileNet-V1 [HZC$^+$17], and we also examine transfer performance for object detection tasks.

Contribution. Our main finding is that sparse models can consistently match the accuracy of the corresponding dense models on transfer tasks. However, this behaviour is impacted by the following factors: *pruning method* (e.g. regularization vs. progressive pruning), *transfer approach* (full vs. linear), *model sparsity* (e.g. moderate 80% vs. high 98% sparsity), and *task type* (e.g. degree of specialization).

Dataset	Linear Finetuning			Full Finetuning		
	Dense	80% Sparse	90% Sparse	Dense	80% Sparse	90% Sparse
Aircraft	49.2 ± 0.1	55.2± 0.2	**56.6 ± 0.1**	83.6 ± 0.4	**84.8 ± 0.2**	**84.9 ± 0.3**
Birds	57.7 ± 0.1	58.4 ± 0.	**58.7 ± 0.**	72.4 ± 0.3	**73.4 ± 0.1**	72.9 ± 0.2
Caltech-101	91.9 ± 0.1	**92.4 ± 0.2**	**92.5 ± 0.1**	**93.5 ± 0.1**	**93.7 ± 0.1**	**93.9 ± 0.3**
Caltech-256	**84.8 ± 0.1**	**84.6 ± 0.1**	84.5 ± 0.1	**86.1 ± 0.1**	85.4 ± 0.2	84.8 ± 0.1
Cars	53.4 ± 0.1	58.6 ± 0.1	**60.5 ± 0.1**	**90.3 ± 0.2**	**90.5 ± 0.2**	90.0 ± 0.2
CIFAR-10	91.2 ± 0.	**91.4 ± 0.**	91.0 ± 0.	**97.4 ± 0.**	97.2 ± 0.1	97.1 ± 0.
CIFAR-100	**74.6 ± 0.1**	**74.7 ± 0.1**	74.3 ± 0.	**85.6 ± 0.2**	85.1 ± 0.1	84.4 ± 0.2
DTD	73.5 ± 0.2	**74.4 ± 0.1**	73.8 ± 0.1	**76.2 ± 0.3**	**75.7 ± 0.5**	**75.5 ± 0.4**
Flowers	91.6 ± 0.1	**93.0 ± 0.**	**93.0 ± 0.1**	95.0 ± 0.1	**96.1 ± 0.1**	**96.1 ± 0.1**
Food-101	73.2 ± 0.	**73.9 ± 0.**	**73.8 ± 0.**	**87.3 ± 0.1**	**87.4 ± 0.1**	**87.3 ± 0.2**
Pets	**92.6 ± 0.1**	**92.5 ± 0.1**	92.0 ± 0.1	**93.4 ± 0.1**	**93.4 ± 0.1**	92.7 ± 0.3
SUN397	60.1 ± 0.	**60.4 ± 0.**	59.8 ± 0.1	**64.8 ± 0.**	64.0 ± 0.	63.0 ± 0.

Table 4.1: Best transfer accuracies at 80% and 90% sparsity for linear and full finetuning, relative to dense transfer. For each downstream task, we present the maximum test accuracy across all sparse methods, highlighting the top accuracy. (We highlight multiple methods when confidence intervals overlap. Results are averaged across five and three trials for linear and full finetuning, respectively.) Note that in all but three cases (all full finetuning), there is at least one sparse model that is competitive with or better than the dense baseline.

We briefly outline our main conclusions, which are also summarized in Figure 4.1 and Table 4.1:

- For *linear finetuning*, sparse models usually match and can slightly outperform dense models, with *regularization-based* methods performing particularly well, even at high sparsities (e.g. 95%).
- For *full finetuning*, which generally provides higher accuracies [KSL19], sparse models are also competitive with dense ones, but transfer accuracy is more tightly correlated with accuracy on the ImageNet pre-training task: consequently, less sparse models (e.g. 80%-90% sparsity) tend to be more accurate than sparser ones. Moreover, in this setting we find that *progressive sparsification* methods consistently produce models with higher transfer accuracy, relative to regularization methods. We provide a first analysis of this effect, linking it to structural properties of the pruned models. In addition, we observe the markedly lower accuracy of lottery-ticket approaches, especially at the higher levels of sparsity, e.g. $\geq 90\%$, required for computational speedups.
- Given the difference in behaviour between linear and full finetuning, we find that there is currently no single "best" pruning method for transfer. However, using existing methods, one can consistently achieve order-of-magnitude ($\sim 90\%$) compression without loss of accuracy. In turn, these compression levels can lead to speedups of more than $3\times$ on sparsity-enabled runtimes. This suggests that sparse transfer may have significant practical potential.

4.2 Background and Related Work

Since we have already described in Chapter 2, Section 2.2.2 the details of each pruning category we consider (progressive sparsification, regularization methods, LTH methods), together with concrete examples, we will now focus on more related work on transfer learning, in general, and sparse transfer learning, in particular.

4.2.1 Transfer Learning and Sparsity

Dense Transfer Learning. A large body of literature has established that, in general, deep learning architectures transfer well to smaller "downstream" tasks, and that full finetuning typically achieves higher accuracy than linear finetuning [KSL19, SIE+20]. However, a recent study [KRJ+22] suggests that this may be inverted on out-of-distribution tasks. These findings extend to related tasks, such as object detection and segmentation [MUK+22]. Kolesnikov et al. [KBZ+20] have focused on factors determining the success of transfer learning, and on developing reliable fine-tuning recipes. This has been further extended by Djolonga et al. [DYT+21], who concluded that increasing the scale of the original model and dataset significantly improves out-of-distribution and transfer performance, despite having marginal impact on the original accuracy. Salman et al. [SIE+20] considered whether *adversarially robust* ImageNet classifiers can outperform standard ones for transfer learning, and find that this can indeed be the case. We complement these studies by examining sparse models and pruning methods.

Sparse Transfer Learning. One of the earliest works to consider transfer performance for pruned models was [MTK+17], with the goal of designing algorithms which allow the pruning of a (dense) convolutional model when transferring on a target task. A similar study was also performed by [SWR20] for language models. By contrast, we focus on the different setting where models have already been sparsified on the *upstream* dataset, and observe higher sparsities than the early study of [MTK+17].

Recent work on sparse transfer learning has focused specifically on models obtained via the "Lottery Ticket Hypothesis" (LTH) approach [FC19], which roughly states that there exist sparsity masks and initializations which allow accurate sparse networks to be trained *from scratch*. There are several works investigating the "transferrability" of models obtained via this procedure for different tasks: for instance, [Meh19] shows that lottery tickets obtained on the CIFAR dataset can transfer well on smaller downstream tasks, while [CFC+20, GMG+21] investigate the applicability of lottery tickets for pre-trained language models (BERT), and object recognition tasks, respectively. Mallya et al. [MDL18] considered the related but different problem of adapting a fixed network to multiple downstream tasks, by learning task-specific masks.

The recent work of [CFC+21] considers the transfer performance of LTH for transfer, proposing a method which we call LTH-T, and finding that this method ensures good downstream accuracy at moderate sparsities (e.g., up to 80%). We consider a similar setting, but investigate a wider array of pruning methods (including LTH-T) and additional transfer datasets. Specifically, we are the first to compare LTH-T to competitive upstream pruning methods. We observe that, on full finetuning, most pruning methods consistently outperform LTH-T in terms of downstream accuracy across sparsity levels, by large margins at high sparsities.

4.3 Methodology

4.3.1 Transfer Learning Setup

Transfer Learning Variants. We consider both *full finetuning*, where the entire set of features is optimized over the downstream dataset, and *linear finetuning*, where only the last layer classifier is finetuned, over sparse models. In the former case, with the exception of

Dataset	Number of Classes	Train/Test Examples	Accuracy Metric
SUN397[XHE+10]	397	19 850 / 19 850	Top-1
FGVC Aircraft[MRK+13]	100	6 667 / 3 333	Mean Per-Class
Birdsnap[BLL+14]	500	32 677 / 8 171	Top-1
Caltech-101[LFP04]	101	3 030 / 5 647	Mean Per-Class
Caltech-256[GHP06]	257	15 420 / 15 187	Mean Per-Class
Stanford Cars[KSDFF13]	196	8 144 / 8 041	Top-1
CIFAR-10[KH+09]	10	50 000 / 10 000	Top-1
CIFAR-100[KH+09]	100	50 000 / 10 000	Top-1
Describable Textures (DTD)[CMK+14]	47	3 760 / 1 880	Top-1
Oxford 102 Flowers[NZ06]	102	2 040 / 6 149	Mean Per-Class
Food-101[BGVG14]	101	75 750 / 25 250	Top-1
Oxford-IIIT Pets[PVZJ12]	37	3 680 / 3 669	Mean Per-Class

Table 4.2: Datasets used as downstream tasks for transfer learning.

the final classification layer and the Batch Normalization (BN) parameters, only the nonzero weights of the original model are optimized, and the mask is kept fixed.

We do not consider from-scratch training and pruning on the downstream task, for two reasons. First, from-scratch training is often less accurate than (dense) transfer learning in the same setting [KSL19, MUK+22]. As our experiments will show, transfer from *sparse models* can often match or even slightly outperform transfer from *dense models*. Second, since training from scratch is typically less accurate than transfer [KSL19], it is unlikely that training and pruning from scratch can outperform sparse transfer from pretrained ImageNet models. To support this claim, we consider image classification on the CIFAR-100 [KH+09] dataset using a WideResNet [ZK16] architecture. Following [PIVA21], the 90% sparse models trained from scratch reach below 80% Top-1 test accuracy. In contrast, finetuning from a ResNet50 backbone pruned on ImageNet using AC/DC and GMP at 90% sparsity reaches a test accuracy of 83.9% and 84.4%, respectively (please see Appendix Table B.3 for the complete results). This example serves to illustrate the significant accuracy gains from using transfer learning with sparse models, as opposed to training sparse models from scratch. Moreover, one practical advantage of transfer learning from sparse models is that it eliminates the need for individual hyper-parameter tuning with respect to compression on the downstream dataset, which can be a costly process.

Downstream tasks and training. We follow [SIE+20] in using the twelve standard transfer benchmark datasets described in Table 4.2; these tasks are a good choice for studying the transfer properties of different models, since they span several domains and sizes. We transfer all parameters of the upstream model except for the last (fully connected) layer, which is adjusted to the number of classes in the downstream task, using Kaiming uniform initialization [HZRS15], and kept dense. It is worth noting that this may slightly change the sparsity of the model, as in some cases the final layer was sparse. As a convention, when discussing sparsity levels, we refer to the *upstream* checkpoint sparsity. For reproducibility, we provide full training hyperparameters in Appendix B.1.

4.3.2 Choice of Sparse Models

Network Architectures. Our study is based on an in-depth analysis of sparse transfer using the ResNet50 architecture [HZRS16]. This architecture has widespread practical adoption, and

has been extensively studied in the context of transfer learning [KSL19, SIE+20]. Importantly, its compressibility has also emerged as a consistent benchmark for methods developed for pruning convolutional neural networks [HABN+21]. We further validate some of our findings on ResNet18, ResNet34 and MobileNet [HZC+17] architectures. In addition, we investigate transfer between two classical object detection tasks, MS COCO [LMB+14] and Pascal VOC [EVGW+10], using variants of the YOLOv3 architecture [RF18].

Sparsification Methods. Following the classification of sparse models proposed earlier into progressive sparsification, regularization and lottery-ticket-hypothesis-based methods, we chose for our study the pruning methods from each category, which provided top ImageNet validation accuracy. Namely, for *progressive sparsification methods*, we use the leading WoodFisher [SA20] and Gradual Magnitude Pruning (GMP) [Hag94, HPTD15, ZG17, GEH19] methods. For *regularization methods*, we consider the leading Soft Threshold Weight Reparametrization (STR) [KRS+20], and Alternating Compression/Decompression (AC/DC) [PIVA21] (presented in Chapter 3) methods. Additionally, we include the "The Rigged Lottery" (RigL) method [EGM+20] with Erdős-Rényi-Kernel (ERK) sparsity distribution. Compared to STR and AC/DC, RigL extends the training schedules on ImageNet by up to 5x, and does *sparse training* for most of the optimization steps. As in Chapter 3, we consider both the standard version or RigL (RigL ERK 1x), and the variant with 5x training iterations (RigL ERK 5x). We will show later in Section 4.4.5 that extending the upstream training time of regularization methods has an important positive impact on transfer. Finally, for *LTH Methods*, we consider the LTH-for-Transfer (which we call LTH-T) method of [CFC+21]; this method precisely matches our setting. In this version, the authors apply the masks obtained through progressive sparsification methods (iterative magnitude pruning [FC19]) directly to the original trained ImageNet dense model (e.g. the standard model provided in the Torchvision library [PGM+19]), and evaluate the transfer accuracy of this masked model through full finetuning on different downstream tasks.

In terms of implementation, when available, we use original sparse PyTorch [PGM+19] checkpoints, and the exact architectures used by the upstream models, as provided by the authors of the methods we consider. However, since the STR and RigL models were trained using label smoothing, which has been shown in [KSL19] to decrease transfer accuracy, we used retrained versions of these models on ImageNet, without label smoothing. The results we discuss in the following sections are for these versions, which indeed perform better, particularly on linear finetuning; we provide an exact comparison regarding the impact of label smoothing in Appendix B.7.1. We manually ported RigL checkpoints from TensorFlow to PyTorch and we were able to obtain close accuracy with the original checkpoints, as it can be seen in Table 4.3 for all ImageNet results.

Lastly, as discussed in Section 2.2.3, we emphasize that there are additional types of pruning methods developed in the literature, for example structured or semi-structured. Our study focuses however on *unstructured* pruning, as these methods are the most studied in the pruning literature, have well established benchmarks, and achieve the best trade-off between accuracy and compression. For completeness, we further include results for full finetuning from models with structured sparsity in Appendix B.5, showing that, given a fixed accuracy level upstream, structured-sparse models tend to underperform unstructured-sparse models for transfer.

4.3.3 Evaluation Metrics

The main quantity of interest is the top-1 validation accuracy on each transfer task, measured for all the pruned models, as well as for the dense baselines. In some cases, we use the mean per-class validation accuracy following the convention for each particular dataset, and we provide these details in Table 4.2. To determine the overall "transfer potential" for each pruning method, we further present the results aggregated over the downstream tasks. Since the datasets we use for transfer learning have varying levels of difficulty, as reflected by the wide range of transfer accuracies, we compute for each downstream task and model the *relative increase in error* over the dense baseline. Specifically, if B is the baseline dense model, then for every downstream task D and sparse model S we define the relative increase in error as $\alpha_{D,S} = \frac{err_{D,S} - err_{D,B}}{err_{D,B}}$, where $err_{D,S}$ is the error corresponding to the top validation accuracy for model S trained on dataset D. For each pruning method and sparsity level, we report the mean and standard error of $\alpha_{D,S}$, computed over all downstream tasks.

We also examine the computational speedup potential of each method, along with its accuracy. For *inference-time* speedups, our findings are in line with previous work, e.g. [EDGS20, SA20, PIVA21]. We will therefore focus on the *training-time* speedup potential in the case of *linear finetuning*, which are usually close to inference-time speedups, as the only difference is the training time of the classifier layer.

We note that Top-1 test accuracy is the standard metric for comparing pruning methods. We also adopt this metric for examining accuracy in the context of transfer, as there is no previous work providing a systematic study on this topic, across multiple pruning methods and sparsity levels. However, we wish to highlight previous work [HCC$^+$19, HMC$^+$20, LBC$^+$21] which examines the impact of pruning on metrics beyond accuracy, such as robustness of pruned models to input perturbations, as well as the impact of pruning on accuracy of specific segments of the data.

4.4 Sparse Transfer on ImageNet

In this section, we describe our main results studying the effects of the choice of the pruning method, together with the target sparsity, on the transferability on different downstream tasks. First, we analyze the impact that the pruning methods considered, at different sparsities, have on the upstream accuracy, on different version of the ImageNet validation set. Then, we study the impact of pruning on both linear and full finetuning. We further investigate potential factors leading to the gap in transfer performance between different pruning methods. Lastly, we show that extending the training time of upstream sparse models can have a substantial positive impact on transfer accuracy.

4.4.1 Validation Accuracy on ImageNet Variants

Setup. We first evaluate the impact of different pruning methods on the ImageNet validation accuracy; this serves as a baseline, and when we further discuss the differences in transfer performance, we will compare them against the accuracy results on the upstream task. Additionally, we examine the accuracy of each pruning method on different versions of the original ImageNet validation set. Namely, we use the ImageNet "reassessed labels" [BHK$^+$20], where the original ImageNet validation images are re-assessed by human annotators, to better capture the diversity of the samples. We also use three different ImageNetV2 validation sets [RRSS19], which contain new images with a similar data distribution, gathered based on the

Sparsity	Method	Original Validation	Reassessed Labels	ImageNetV2 (Average)
0%	Dense	76.8%	83.1%	72%
80%	AC/DC	76.2%	82.9%	71.8%
	STR	75.5%	81.9%	70.3%
	WoodFisher	**76.7%**	**83.2%**	**72.3%**
	GMP	76.4%	82.9%	71.6%
	RigL ERK 1x	74.8%	81.3%	70.2%
	RigL ERK 5x	75.8%	81.6%	70.6%
90%	AC/DC	75.2%	82.2%	70.6%
	STR	74.0%	80.9%	69.1%
	WoodFisher	75.1%	**82.4%**	**71.1%**
	GMP	74.7%	81.6%	70.1%
	RigL ERK 1x	73.2%	80.0%	67.9%
	RigL ERK 5x	**75.7%**	81.9%	70.6%
95%	AC/DC	73.1%	80.4%	68.6%
	STR	70.4%	77.9%	66.0%
	WoodFisher	72.0%	79.8%	67.6%
	RigL ERK 1x	70.1%	77.5%	65.5%
	RigL ERK 5x	**74.0%**	**80.8%**	**69.0%**

Table 4.3: Accuracy of the pruning methods we use, at different sparsity levels, evaluated on different ImageNet validation sets.

frequency by which an image was selected by its annotators; in our results, we report the average accuracy across these three variants.

Discussion. Through our results presented in Table 4.3 we observe that RigL ERK 5x outperforms all methods on the original validation set at 90% and 95% sparsity, followed by AC/DC, GMP and WoodFisher. At 80% sparsity, WoodFisher has the best original validation accuracy, followed closely by GMP and AC/DC. However, despite the gap in original validation accuracy between RigL ERK 5x and other methods, the results on new variants of the validation set still reveal some interesting patterns. For example, WoodFisher outperforms all methods at 80% and 90% sparsity on the reassessed labels, followed closely by AC/DC. This is true also for ImageNetV2, where WoodFisher outperforms all methods at 80% and 90% sparsity. At 95% sparsity, however, RigL ERK 5x outperforms all methods considered, including on the reassessed labels and ImageNetV2, and is followed by AC/DC. Generally, the accuracies on the reassessed labels and ImageNetV2 correlate well with those on the original images, which suggests that top performing methods can "extrapolate" well. However, we note that the ImageNet accuracies for some of the pruning methods (AC/DC, WoodFisher or GMP) are fairly close at 80% and 90% sparsities; this further motivates our study regarding the differences between pruning methods beyond model accuracy on the upstream task.

4.4.2 Linear Finetuning

Setup. In this section we study the transfer performance of different types of pruning methods in the scenario where only the linear classifier "on top" of a fixed representation is trained on the downstream task. Specifically, we study the simple setup where the features prior to the final classification layer of the pre-trained model are extracted for all samples in the transfer dataset and stored into memory for use when training the downstream linear classifier. Although

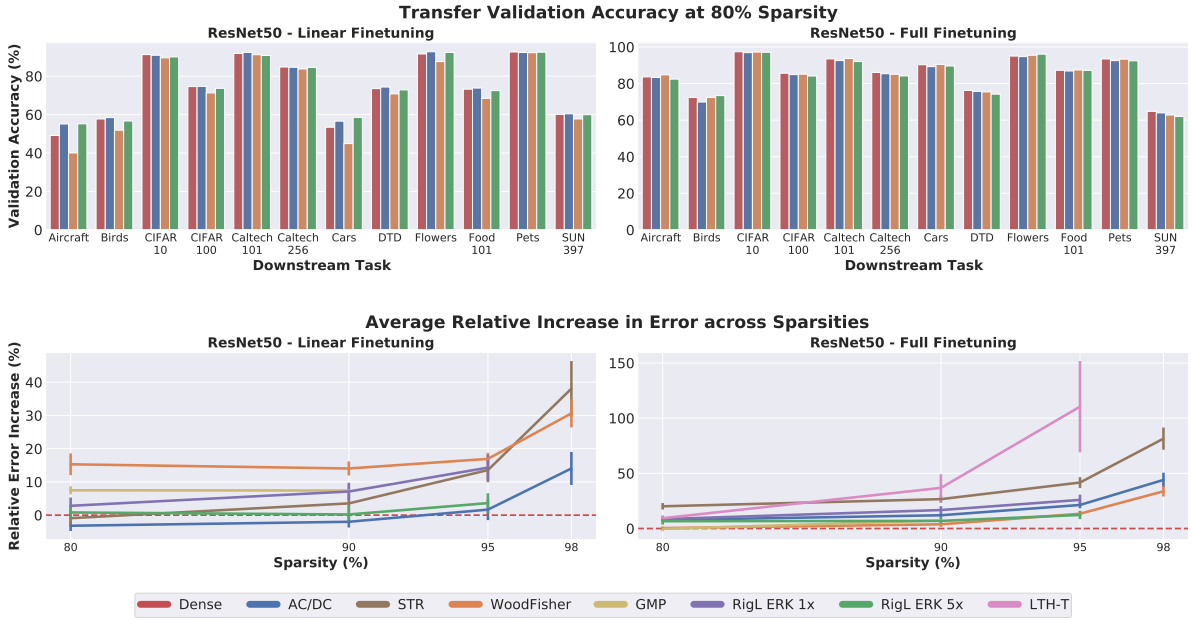


Figure 4.2: (top row) Accuracy for selected pruning strategies at 80% sparsity. (bottom row) Average increase in test error relative to the dense baseline; lower values are better. Best viewed in color.

this approach typically results in lower accuracy relative to full finetuning [KSL19, SIE+20], it has significant practical advantages. Specifically, the features can be precomputed, which eliminates the forward passes through the pretrained network. In this setup, we do not apply any data augmentation on the transfer samples and we use the Batch Normalization statistics of the pretrained network on ImageNet.

We optimize the linear classifier using SGD with momentum, weight decay and learning rate annealing, following [SIE+20]. (The results are typically well-correlated with those obtained when using data augmentation during training, or using different optimizers [KSL19]). In Section 4.5, we show that training speed-ups can also be obtained in an online learning setup, where new samples are executed through the backbone network, by taking advantage of the backbone sparsity.

Results. The results for linear finetuning are shown in Figure 4.2, and Appendix Table B.2. We exclude the LTH-T method from this analysis, as it is designed for full finetuning, and its transfer accuracy in the linear scenario is indeed very low (see Appendix Table B.2). We believe one reason for this could be the fact that the sparse masks are directly applied to the pretrained ImageNet model; this would result in a discrepancy in the distributions for the activations between the dense and sparse model, for example in case of Batch Normalization layers, which are not subsequently calibrated; this discrepancy would affect the feature representation used in linear finetuning, since in our experiments we use the Batch Normalization statistics pre-computed on ImageNet.

Overall, the results clearly show that the choice of pruning strategy on the upstream task can result in significant differences in performance on downstream tasks. These differences are more apparent for specialized downstream tasks, with fine-grained classes. For example, consider Aircraft, where for 80% sparse models we see a 15% gap in top-1 test accuracy between the best-performing sparse models (AC/DC and RigL, 55%) and the worst-performing one (WoodFisher, 40%).

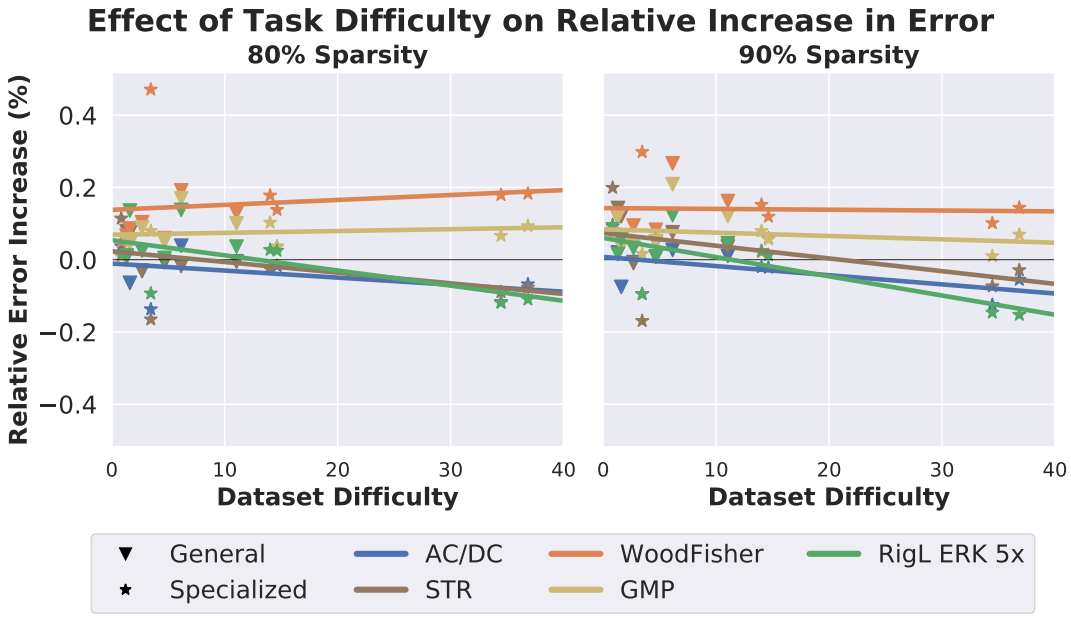


Figure 4.3: Effect of task difficulty on various pruning strategies for transfer with linear finetuning. Best viewed in color.

Impact of Task Difficulty. Following the previous observation, we study the correlation between the downstream task difficulty and relative increase in error for different pruning strategies. For this purpose, we use the difference in Top-1 validation accuracy between full and linear finetuning on the dense backbone as a proxy for the difficulty of a downstream task. Intuitively, a small gap between full and linear finetuning would suggest that the upstream features are directly transferable, and thus the downstream task can be considered "easy". Conversely, a large gap would indicate that the pre-trained features are not enough to capture the internal representation of the data, making the downstream task more "difficult". Additionally, we categorize the downstream tasks into *general* (Caltech-101/256, CIFAR-10/100, DTD, SUN397) vs. *specialized* (Aircraft, Birds, Cars, Flowers, Food-101, Pets); this is similar to previous work [KSL19]. Figure 4.3 suggests that specialized datasets tend to have higher difficulty scores.

Following this definition and categorization, we measure, for each pruning strategy, the relative error increase over the dense model against the task difficulty. Figure 4.3 shows the behavior for all pruning methods considered at 80% and 90% sparsity. Interestingly, we observe a trend for *regularization methods* (AC/DC, STR, RigL) to improve over the dense baseline with increased task difficulty, which is more apparent at higher sparsity (90%). In contrast, progressive sparsification methods (GMP, WoodFisher) do not show a similar behavior. This suggests that regularization pruning methods are a better choice for linear transfer (sometimes even surpassing the dense performance) when the downstream task is more specialized or more difficult.

Another particularity of linear finetuning from sparse models is that the sparsity level is not highly correlated with the performance on the downstream tasks. This is apparent, for example, for AC/DC and RigL, where, despite the 1-2% gap in ImageNet accuracy between the 80% and 90% sparse models, the relative error with respect to the dense baseline stays quite flat. A similar trend can be observed for other pruning methods as well. However, extremely sparse models (98%) tend to perform worse, probably due to feature removal and degradation.

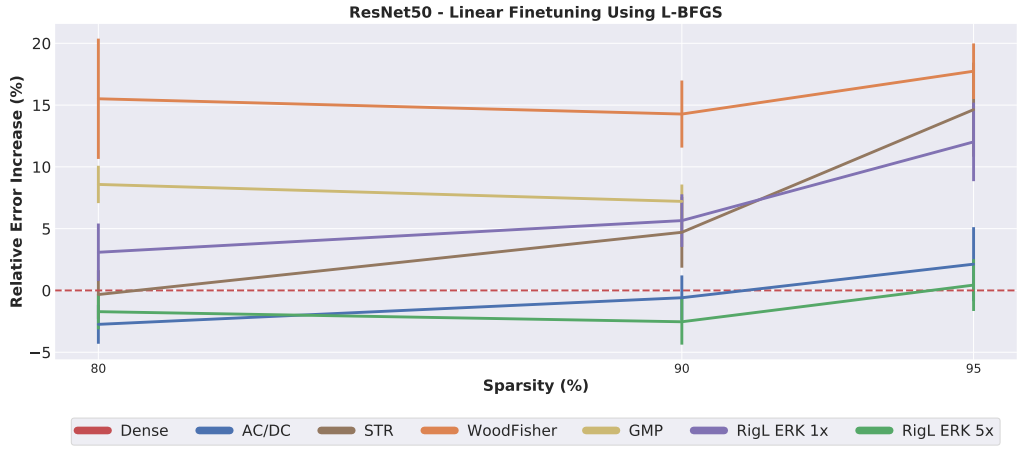


Figure 4.4: (ResNet50 Linear Finetuning) Average relative increase in error across tasks when performing linear finetuning using the L-BFGS optimizer.

Finetuning with a Different Optimizer. To further validate our findings, we test the impact that the choice of the optimizer has on the linear finetuning results. Similar to [KSL19], we perform linear finetuning using the L-BFGS optimizer [LN89], with L_2 regularization. The value of the L_2 hyperparameter is tuned individually on each downstream task, using a validation set. Overall, we observed a slight improvement in the final test accuracies across tasks, compared to SGD-finetuning, as can be seen in the complete results presented in Appendix Table B.1. When examining also the relative increase in error, presented in Figure 4.4, we observe that the similar trend holds: sparse regularization methods outperform progressive sparsification methods. However, while AC/DC performs slightly better compared to the dense baseline at 80% and 90% sparsity, it is outperformed by RigL 5x at 90% and 95% sparsity. Moreover, we observe a slight improvement in the results for RigL 5x at 90% sparsity, compared to 80%, which is consistent with the results obtained when using SGD.

Conclusion. In summary, we observe that 1) some sparsification methods can consistently match or even sometimes outperform dense models; 2) there is a correlation between transfer performance for regularization-based methods and downstream task difficulty; and 3) higher sparsity is not necessarily a disadvantage for transfer performance.

4.4.3 Full Finetuning

Setup. We now consider the *full finetuning* scenario. Here, we re-initialize the final classification layer and keep it dense, then finetune the unpruned weights so that the network is sparse throughout training, with a fixed sparse topology.

Results. We summarize the results in Figure 4.2, and provide the detailed numbers in Appendix Table B.3. Similar to linear finetuning, we see substantial performance variations among pruning strategies when transferred to the downstream tasks. Typically, progressive sparsification methods (WoodFisher, GMP) tend to transfer better than regularization and lottery ticket methods. The differences in test accuracy, measured at the same sparsity level, are typically small, on the order of 1–3%; the exception is LTH-T, which is competitive at low sparsity (80%), but incurs severe accuracy drops at sparsities $\geq 90\%$.

In contrast to linear finetuning, we see a consistent trend of decreasing quality with increased sparsity. This is not surprising, since full finetuning can take advantage of the additional

parameters available in denser models to better fit the downstream data. Nevertheless, *progressive sparsification* methods (GMP and WoodFisher) result in downstream performance nearly on par with dense models at 80% and 90% sparsity. These methods show better performance than regularization-based methods (AC/DC, STR, and RigL), a direct reversal of the results of linear finetuning.

For specific downstream tasks, however, there is considerable variability—while WoodFisher and GMP are consistently the top or near-top performing models across all tasks, other methods show considerable task dependence. For instance, while AC/DC is the top performing method across different sparsities for three of the twelve tasks (SUN397, Caltech-256, and DTD), it shows a considerable gap compared to the best-performing methods on Aircraft, Cars, and CIFAR-10. Generally, STR performs worse on full finetuning, compared to other regularization methods. Furthermore, RigL ERK 1x performs roughly on par with AC/DC, despite having a lower validation accuracy on ImageNet; however, the extended training of RigL ERK 5x gives the tansfer accuracies a considerable boost, putting RigL ERK 5x almost on par with WoodFisher, especially at higher sparsities. Finally, LTH-T shows fairly competitive performance at 80% sparsity, but its transfer accuracy declines dramatically on six of the twelve datasets (SUN397, Caltech-101, Caltech-256, DTD, Flowers, and Pets) as sparsity increases. Since the LTH-T model relies mainly on transferring the sparsity mask across tasks, this suggests that the additional information present in the *weights*, leveraged by other methods, may be beneficial.

Conclusion. In sum, if the goal is to perform full finetuning on downstream tasks, then *progressive sparsification* methods are a good choice. They consistently outperform regularization methods across a wide range of tasks, and offer comparable performance to the dense backbone at 80% and 90% sparsity.

4.4.4 Expressivity of Different Sparse Models

Pruned Filters (%)	AC/DC	WoodFisher	GMP	STR	RigL ERK 1x	RigL ERK 5x
80%	**2.9%**	0.9%	1.6%	0.5%	0.2%	0.6%
90%	**8.5%**	2.0%	2.8%	2.0%	1.2%	2.7%
95%	**18%**	3.0%	-	6.0%	4.3%	9.1%

Table 4.4: Percentage of convolutional filters that are completely masked out, for different pruning methods on ResNet50, at different sparsity levels. AC/DC has significantly more pruned filters.

The results of the last two sections show an intriguing performance gap between pruning methods, depending on the transfer approach. We would like to determine possible factors responsible for these differences. Therefore, to investigate further, we examine the sparse structure of the resulting pruned models, by measuring the percentage of convolutional filters that are completely pruned away during the training phase of the sparse ResNet50 backbones on the original ImageNet dataset. The results in Table 4.4 show the differences in the number of zeroed-out channels among pruning methods. We observe that AC/DC has a large number of channels that are fully removed during ImageNet training and pruning, on average 2-4 more at 80% and 90% sparsity, compared to other models; this results in fewer features that can be trained during full finetuning. By contrast, the sparsity in GMP and WoodFisher is less structured and thus can express additional features, which can be leveraged during finetuning.

To further test this hypothesis, we perform full finetuning from a ResNet50 model pruned with structured sparsity, using the ℓ_1 norm of the convolutional filters as a pruning criterion. Thus, by definition, models with structured sparsity have a considerable number of filters removed, compared to the baseline. The model we consider achieves around 2x inference speed-up compared to the dense baseline, which is a similar speed-up to a model pruned to 90% global sparsity, and has 75.7% ImageNet validation accuracy. Interestingly, the accuracy for full finetuning when using this model tends to be lower than that of the best performing 90% sparse model (e.g. WoodFisher). We fully illustrate this effect in Appendix B.5, where we present the result on each downstream task.

In the case of linear finetuning, we hypothesize that the accuracy reversal in favor of AC/DC can be attributed to a regularizing effect, which produces more "robust" features. The same effect appears to be present in RigL ERK 5x at 95% sparsity, which also has significantly many fully-pruned filters.

4.4.5 The Effect of Extended Training on Sparse Transfer

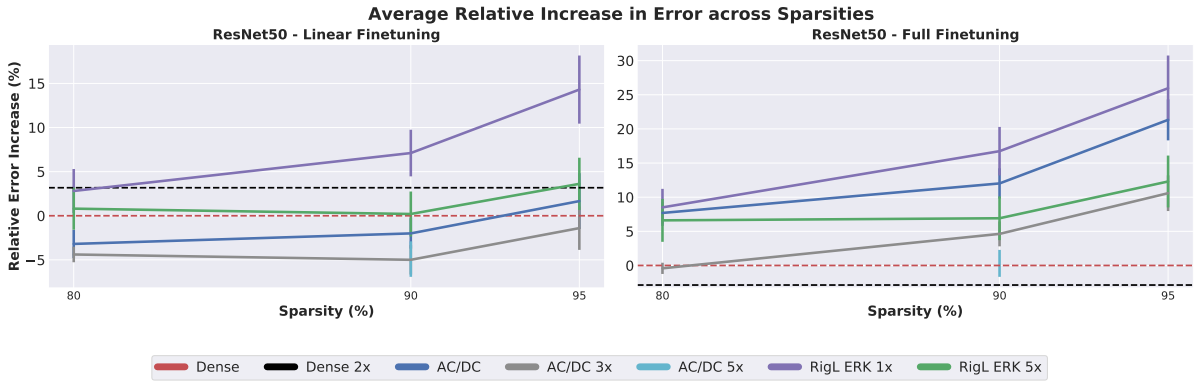


Figure 4.5: (ResNet50) Overall performance for extended-time runs of AC/DC and RigL. The AC/DC 5x was only done for 90% sparsity; its performance is almost the same as AC/DC 3x for linear finetuning and as dense for full finetuning.

In this section we examine the effects of extended training time on the transfer accuracy, for both the dense baseline and for sparse regularization methods.

Setup. As noted in the previous sections discussing linear and full finetuning results, RigL models trained for extended number of epochs (5x the standard value) show substantial improvements in the full finetuning performance, bridging the gap towards the dense transfer accuracy. One obvious explanation would be the improved accuracy on the upstream task, which has been shown to generally correlate well with the full transfer accuracy, as mentioned in Section 4.4.3. These observations, however, motivate us to further investigate whether the phenomenon of improved transfer accuracy with extended upstream training time is more universal across pruning methods. For this, we use the AC/DC checkpoints discussed in Chapter 3, which were also trained for extended number of epochs. Namely, we consider AC/DC trained for 300 of epochs on ImageNet/ResNet50 (AC/DC 3x) at 80%, 90% and 95% sparsity, and AC/DC trained for 500 epochs (AC/DC 5x) at 90% sparsity. Please see the Section 3.4.4 from Chapter 3 for more details regarding the performance of these models on ImageNet. For completeness, we also train the dense baseline model for 200 epochs, which is twice more than the standard value. If in the case of sparse training with AC/DC or RigL

the improvements from prolonged training were substantial, for the dense baseline we only observed a minor improvement of 0.3% on the ImageNet validation accuracy, from 76.8% to 77.1%. This is not a surprising behavior related to training dense models, and has been documented in other works as well, for example [FKMN21].

Results. Interestingly, the improvements in upstream accuracy from extending the training time of sparse regularization methods also translate on improved transfer accuracy, for both linear and full finetuning. We present the results showing the average relative increase in error across sparsities for extended training in Figure 4.5, as well as the accuracies of the considered methods on each task in Appendix Tables B.4 and B.5.

In the case of linear finetuning, we observe that extending the training time for the dense model resulted in a surprising decrease in transfer performance. However, the reversal is true for sparse regularization methods. Notably, AC/DC 3x at all sparsities considered (80%, 90% and 95%), as well as AC/DC 5x at 90% sparsity outperform the dense model, and, interestingly, the transfer performance slightly improves at 95% sparsity.

We observe substantial improvements also for full finetuning. In contrast to the linear finetuning case, here the dense 2x model slightly outperforms the baseline. Furthermore the transfer performance of AC/DC 3x at 80% sparsity is on-par with dense, and is close to the overall transfer performance of WoodFisher, which was previously the best performing pruning method on the full finetuning task. Notably, the 90% sparse AC/DC 5x model performs on-par with the dense baseline, which makes it the best performing sparse model for full finetuning.

Conclusion. Our results show that extending the training time on the upstream task can lead to substantial improvements in transfer performance for sparse regularization methods, in both the linear and full transfer scenarios. While this phenomenon was initially observed for RigL models, it is further confirmed for AC/DC. Notably, with extended training time we obtain 90% sparse models (AC/DC 5x) with the same full transfer performance as the dense baseline, and we surpass the dense model at all sparsity levels for linear finetuning.

4.5 Practical Implications of Sparse Transfer

One of the main benefits of sparse models is that they can provide *inference speed-ups* when executed on sparsity-aware runtimes [EDGS20, SA20, PIVA21, KKG+20]. For linear finetuning, this can also imply *training time* speed-ups, since the sparse backbone is fixed, and only used for inference. We illustrate this in an "online learning" setting, where training samples arrive dynamically at the device. We first compute the corresponding features using the sparse backbone. Then, we use these features to train the linear classifier, which implies that the forward pass can benefit from speed-ups due to sparsity.

To demonstrate that we can obtain practical training-time speed-ups, and to measure the quality of the resulting finetuned models , we integrated the freely-available sparsity-aware DeepSparse CPU inference engine [KKG+20, Dee21] into our PyTorch pipeline. Specifically, we use sparse inference for online feature extraction. In this case, the model is completely fixed, including the Batch Normalization statistics, which are used in evaluation mode, but we allow random augmentations on the input samples. We report overall training speedup, in terms of average training time per epoch on the downstream task, as a fraction of the average training time using the dense baseline. Apart from the use of data augmentation, hyperparameters are

identical to the linear finetuning experiment in Section 4.4.2. For reproducibility, we execute on an Intel E5-1650 CPU with 12 cores, which would be similar in performance to a recent laptop CPU. The speed-ups we report are proportional to the inference speed-ups of the respective sparse backbone models. The only difference is the cost of optimizing the last layer, which varies in size with the number of classes.

Figure 4.6 shows results on four downstream tasks, Pets, Flowers, DTD and Caltech-101, where the backbone ResNet50 models have 90% sparsity. We report the training speed-up vs. the difference in test accuracy, compared to the dense baseline. We note that there are differences between the results in Figure 4.6 and those discussed in Section 4.4.2, which arise from the use of data augmentation in the current setup.

Furthermore, we show speed-up numbers for additional sparsity levels (80% and 95%), in Table 4.5; these numbers representing the average training time per epoch are computed on the Caltech-101 dataset, but the speed-up factor should be similar for other datasets as well, since the training time per batch is almost proportional to the inference through the backbone network. These results show that *using sparse backbones can reduce training time by 2-4x for linear transfer, without negative impact on validation accuracy*, which is encouraging for transfer applications on low-energy devices.

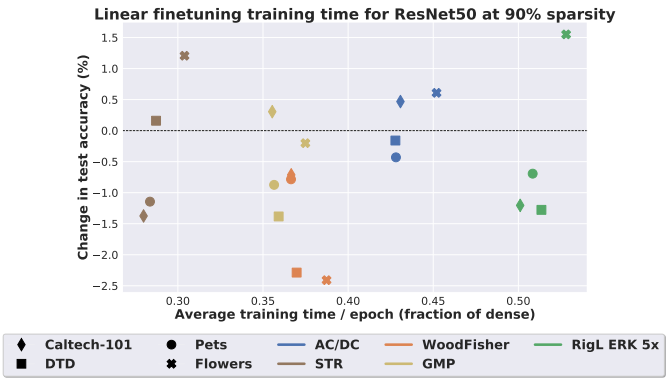


Figure 4.6: Average epoch time vs. gap in test accuracy, compared to the dense baseline. Results are shown for four different downstream tasks, using linear finetuning from ResNet50 90% sparse models. Lower is better; best viewed in color.

Sparsity	STR	GMP	WoodFisher	AC/DC	RigL 5x
80%	0.44×	0.50×	0.53×	0.60×	0.71×
90%	0.28×	0.36×	0.37×	0.43×	0.50×
95%	0.22×	N /A	0.28×	0.32×	0.36×

Table 4.5: Average training time per epoch for linear finetuning using sparse models, as a fraction of the time per epoch required for the dense backbone. The numbers shown are computed on the Caltech-101 dataset.

4.6 Extensions to Different Models and Tasks

In addition to our in-depth study of sparse transfer on ResNet50 models pretrained on ImageNet, we also extend our findings to other architectures (ResNet18/34 and MobileNet) or other tasks, such as object detection using YOLO models, or knowledge distillation [HVD15].

ResNet18/34 and MobileNet Experiments. We extended a subset of the sparse transfer experiments on ResNet18, ResNet34 and MobileNetV1 [HZC+17] models trained on ImageNet. Namely, on ResNet18 and ResNet34 we compare the linear transfer performance between the dense baseline, regularization and progressive pruning methods. For regularization-based methods, we use AC/DC, while for progressive sparsification we use GMP models; for each we consider 80% and 90% sparsity levels. The results on ResNet18 and ResNet34 largely confirm our conclusions from ResNet50; namely, regularization based methods match or outperform the dense baseline, on linear finetuning tasks. These results are presented in detail in Appendix B.3.

Furthermore, we also extend our study on the MobileNetV1 architecture and use publicly available checkpoints, such as AC/DC, M-FAC [FKA21] or STR. In this case, we observe that sparse models can match the dense baseline transfer performance only at lower sparsities (up to 75%), for both linear and full finetuning; we believe that this is due to the lower parameter count, which makes MobileNet harder to prune. Interestingly, we also observe a reverse effect between regularization and progressive sparsification methods; namely, AC/DC models tend to perform slightly better than M-FAC on full finetuning, and slightly worse on linear finetuning. We present our results in detail in Appendix B.4.

Sparse Transfer using YOLO. We also examined transfer performance between YOLO V3 [RF18] and YOLO "V5" [Ult21] models for object detection, trained and pruned on the COCO dataset [LMB+14], which are then transferred to the VOC dataset [EVGW+10] using full finetuning. Table 4.6 presents results in terms of mean Average Precision (mAP@0.5). Results show a strong correlation between accuracy on the original COCO dataset and that on VOC, confirming our claims. We observed similar trends in a segmentation setup, which we cover in Appendix B.6.

Architecture Pruning	YOLOv3 90% Sparsity	YOLOv5S 75% Sparsity	YOLOv5L 85% Sparsity
COCO Dense	64.2	55.6	65.4
COCO Pruned	62.4	53.4	64.3
VOC Dense Transfer	86.0	83.73	90.0
VOC Pruned Transfer	84.0	81.72	89.35

Table 4.6: Accuracies for Sparse Transfer from COCO to VOC.

Distillation from Sparse Teachers We further investigate a different aspect of transfer learning, namely knowledge distillation, which was introduced in [HVD15] as a method for enhancing the performance of smaller models, called students, by incorporating during training the information learned by a larger, more specialized model, called teacher. The intuition for using sparse models as teachers is based on the assumption [HCC+19] that they usually are less confident than their dense counterparts, and as such could offer additional information regarding the other classes. Moreover, we are also motivated by our linear finetuning experiments, which suggest that sparse models may provide superior representations relative to dense ones.

Baseline	Dense KD	AC/DC 80%	WoodFisher 80%	AC/DC 90%	WoodFisher 90%
73.83%	74.42%	74.64%	74.63%	74.19%	74.44%

Table 4.7: Top-1 validation accuracy on ResNet34 trained on ImageNet, when distilling from dense or sparse teachers.

We experimented with distilling a ResNet34 model trained on ImageNet, from a ResNet50 teacher. For the teacher, we use the WoodFisher and AC/DC models, at 80% and 90% sparsity. Our results presented in Table 4.7, show that indeed sparse teachers can have similar performance to the dense teacher, suggesting that differences in accuracy between the sparse and dense teachers do not affect distillation. Nonetheless, these results are encouraging, since using sparse teachers could also reduce distillation overhead due to faster inference, for

example by using the DeepSparse library [KKG+20]. This is particularly important, as previous studies, such as [BZR+22], have shown that distillation benefits when the teacher sees the exact same samples as the student, up to data augmentation fluctuations.

4.7 Conclusions and Future Work

In this work, we performed an in-depth study of the transfer performance of sparse convolutional neural networks, and showed that pruning methods with similar accuracy on ImageNet can have surprisingly disparate Top-1 accuracy when used for transfer learning. In particular, regularization-based methods perform best for linear finetuning; conversely, progressive sparsification methods such as GMP and WoodFisher tend to work best when full finetuning is used.

Our empirical study also aims to potentially reveal some analytical insights based on the practical results obtained: we hypothesize that the differences in transfer performance between progressive sparsification and regularization methods could be related to the structure of the sparsity distributions learned through each method; for example, regularization methods seem to induce more structured sparsity, which might restrict the expressivity of the model in case of full finetuning, but might have a regularizing effect in the linear finetuning setup. An interesting direction for future work would be to analyze the sparse features representations in terms of their generalization properties.

Additionally, we would like to point out some limitations of our study. The first one would be that it only investigates accuracy as a measure of performance for transfer learning tasks. We note that additional research is needed towards designing pruning strategies with good performance across *both* linear and full finetuning, and towards considering metrics past Top-1 accuracy, such as bias and robustness. Another limitation is that we considered a (standard) fixed set of transfer datasets; our study should be extended to other, more complex transfer learning scenarios, such as distributional shift [KSM+21]. Further investigation could also systematically examine other types of compression, such as quantization and structured pruning, potentially in conjunction with unstructured pruning, which was the focus of our current study. Other interesting areas for future work would be understanding the performance gap between full finetuning and linear finetuning, and realizing training speedups for sparse full finetuning, by taking advantage of the fixed sparsity in the trained model.

CHAPTER 5

Training Prunable Models Through Compression-Aware Minimization

We have already seen in Chapter 3 that sparse models can have a very good performance in terms of accuracy, while providing computational saving during both training and inference. Moreover, through our study presented in Chapter 4, we concluded that sparse models enjoy additional interesting generalization properties, such as good transferability to tasks different from the original ones used during their training.

However, we note that well-performing sparse models still usually require re-training or finetuning for each sparsity target individually. Since this process can be quite costly, ideally we would like a method that can train an accurate model that can be later on pruned to multiple sparsity levels, without any additional finetuning. In this chapter, we present a method that can achieve this goal.

5.1 Motivation and Outline

We propose *Compression-Aware Minimization (CrAM)*, a method for training neural networks, which results in models that are easily compressible *one-shot*, while still being highly-accurate. Specifically, CrAM enables training a single (dense) model, which can later be compressed to different target levels, with minimal or no recalibration. Such flexibility is desirable, as models can be trained once, and then deployed on multiple devices, with different specifications. Having a single model that can be easily configured to meet the computational requirements of a specific device can both reduce the overall computational cost, and also allow easier customization to individual devices.

CrAM is loosely-inspired by the recently-introduced sharpness-aware minimizer (SAM) [FKMN21], which trains models that potentially converge to flatter minima, leading to better generalization compared to SGD-type baselines, by biasing the process towards minima of *uniformly low loss*. Multiple subsequent works have investigated and improved upon the original SAM algorithm, by either obtaining better generalization [KKPC21], or by reducing the computational costs of SAM training [LZB20, DYF+22]. We are the first to carry over this idea to the task of obtaining *compressible* models. Roughly speaking, CrAM works by optimizing not against the original "dense" model, but over a compression projection applied to the intermediate model iterate, at every optimization step. Thus, the CrAM update aims to bias optimization

towards iterates that have both *low loss* and are *robust under one-shot compression*. Similarly to SAM, CrAM is simple to implement as part of a regular training loop and has a single scaling hyper-parameter, for which we provide a well-performing default value. We detail the CrAM algorithm and provide a theoretical motivation leveraging fundamental results in robust optimization [Dan12] in Section 5.3.

To complement our algorithmic contribution, we perform an extensive experimental analysis of CrAM. We mainly focus on compression via weight pruning, but we also show that CrAM is compatible with weight quantization. Generally, CrAM models trained on large-scale image classification or language modelling tasks can improve over the dense baseline performance, while being very robust to one-shot pruning, at different sparsity levels. For image classification, CrAM can train a highly-accurate dense ResNet50 model on ImageNet, that can be pruned *in one-shot* to 80% and 90% sparsity, and is competitive in terms of accuracy relative to state-of-the-art *gradual pruning methods*, following an inexpensive Batch Normalization re-tuning step on a small calibration set.

Moreover, we show that full CrAM training is not necessary for good performance: specifically, a short CrAM finetuning period is sufficient to substantially improve one-shot pruning accuracy. For instance, we used CrAM to transfer the standard BERT-base model [DCLT19] on the SQuADv1.1 question-answering task [RZLL16], and obtained models that are both more accurate and more compressible than those obtained with standard optimizers, such as Adam [KB15] or SAM [FKMN21]. In addition, we noticed that a short (≤ 2 epochs) finetuning of the *sparse* model can provide substantial additional improvements: on the above task, the 80%-sparse CrAM finetuned model reaches higher accuracy than the highly-competitive gradual pruning methods PLATON [ZZL+22] and Movement Pruning [SWR20], at a fraction of the training budget.

Further, CrAM lends itself to several extensions: it can be used with different layer-wise sparsity distributions, semi-structured N:M sparsity patterns, and one-shot pruning techniques. Sparse CrAM models can be successfully used for sparse transfer learning, where they can perform better on a wide range of "downstream" target tasks, even when compared to pruning methods which adapt to the downstream task [CFC+21]. Lastly, we also provide evidence that the CrAM update can produce models that are robust to quantization.

Similar to SAM [FKMN21], one limitation of our method is the added computational cost, as it requires an additional backwards pass for the model perturbation. This can be addressed by only performing limited finetuning via CrAM instead of full retraining, or by only performing a regular optimization step for a fraction of the time, both of which we show to have a limited impact on accuracy. Moreover, our approach is also compatible with efficient SAM-type updates [LZB20, DYF+22]. We also provide a well-performing variant of CrAM that uses sparse gradients, which could be leveraged by frameworks with support for sparse back-propagation.

5.2 Related Work

We describe in this section some of the recent research directions that have inspired the development of our method, together with existing literature focused on solving similar problems.

Sharpness-Aware Minimization (SAM). The recently introduced SAM optimizer [FKMN21] aims to improve the generalization of deep neural networks, by encouraging the minimization

of loss sharpness; this in turn should lead to flatter local minima, with better generalization properties. Similar to [KMN+17], sharpness is defined as the maximum difference in loss achieved around a bounded-norm perturbation of the parameters; the SAM method proposed in [FKMN21] optimizes the following loss function: $L^{\text{SAM}}(\boldsymbol{\theta}) = \max_{\|\boldsymbol{\delta}\| \leq \rho} L(\boldsymbol{\theta} + \boldsymbol{\delta})$. The perturbation $\boldsymbol{\delta}$ can be approximated as $\rho \cdot \nabla L(\boldsymbol{\theta}) / \|\nabla L(\boldsymbol{\theta})\|$, and the gradient of $L^{\text{SAM}}(\boldsymbol{\theta})$ is approximated using a straight through estimator as $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\boldsymbol{\delta}}$. Subsequently, this gradient is used to optimize the parameters $\boldsymbol{\theta}$ of the model, through SGD. The authors of [FKMN21] show that SAM-trained models have higher validation accuracy compared to vanilla SGD-type baselines, and their performance continues to improve with prolonged training; this suggests that SAM models are less prone to overfitting. Moreover, the authors show that SAM models can also be successfully used for transfer learning. One important drawback of SAM is its computational overhead, as it requires twice as many forward-backward passes through the network. Subsequent work has focused on reducing computational cost by, for example, reducing the frequency of the extra gradient steps [LMC+22], computing the perturbations on a subset of the parameters [DYF+22], or by proposing a new trajectory loss to replace the sharpness definition [DZF+22]. We draw inspiration from properties of the initial SAM method proposed by [FKMN21]. Instead of attempting to minimize the maximum local increase loss (sharpness), our goal is to minimize the maximum local increase in loss due to compression.

Training prunable networks. The increasing scale of deep neural networks have made their deployment to edge devices dependent on compression techniques, such as quantization and/or pruning. While post-training quantization can be an efficient and successful technique for quantizing models without any retraining [FA22a], in the case of pruning the gold standard is still training a separate model for every target sparsity level [ZG17, SA20, EGM+20, PIVA21]; the latter can be an expensive procedure, which would still rely on powerful computational resources to obtain the sparse models in the first place. A potential solution would be training a single dense model, which either contains multiple smaller ones that can be easily deployed, or which is itself *prunable* at multiple sparsity levels, without additional retraining. For example, the "once-for-all" (OFA) framework [CGW+19] can train a large network that contains multiple specialized sub-nets, adapted to different resource constraint devices. However, obtaining the large OFA network is extremely expensive, and requires intensive finetuning to ensure a good performance for the sub-nets. A similar idea that also requires extensive finetuning has been explored for automatic speech recognition [WZL+21].

An orthogonal direction is to obtain "slimmable neural networks" [YYX+19, YH19b, YH19a], by training a single model that can be executed at different widths; this is usually achieved by performing multiple backpropagations using all the predefined widths, at each optimization step, and by carefully considering the Batch Normalization layers. Related to one-shot pruning, Only Train Once (OTO) [CJD+21] has been proposed as a framework for structured pruning, to train a large model that is easily slimmable one-shot. While we obtain better results than OTO for the same sparsity level, the two methods are not directly comparable, since we focus on *unstructured sparsity*. Morover, CrAM modifies the optimization step such that the resulting dense model is both highly accurate, and robust to post-training one-shot pruning, without retraining.

Our work is more closely related to [MLC+22, ZSP22], which propose leveraging Stochastic Frank-Wolfe (SFW) [RSPS16] to encourage the weights to lie in a convex hull spanned by sparse vectors; this would make the model prunable one-shot, without any finetuning. The methods proposed in [MLC+22, ZSP22] result in highly-prunable models on relatively-small tasks; specifically, their experimental analysis is limited to image classification on small datasets and architectures with many redundancies (e.g. VGG-16 [SZ14] on CIFAR-10). CrAM is able

to match or outperform these methods in the same setting: for instance, CrAM can prune VGG-16 trained on CIFAR-10 in one-shot to 95% sparsity without accuracy loss, outperforming SFW by more than 2% Top-1 acuracy. More importantly, we show that CrAM produces models compressible in one-shot at both ImageNet scale and BERT language modeling scale. Remarkably, with one-shot pruning CrAM can offer competitive performance to *gradual pruning* methods, whether they are designed for CNNs [KRS+20, LSB+20] or for language modelling [SWR20, ZZL+22].

5.3 The Compression-Aware Minimizer (CrAM)

5.3.1 Method Description

We now give an overview of our method, together with the corresponding algorithm and generalizations. One of the main goals of CrAM is to train models that are "compressible" in one-shot, following training, via sparsity or quantization. In what follows, we denote C a compression operator, for example Top-K, where only the highest K absolute values of a tensor are kept, while the rest are set to 0. We say that a model is easily compressible if small perturbations do not affect its performance after compression. To enforce this during training, we optimize against the perturbation of the dense model which has the most impact on the compressed model. We want to minimize the "compression-aware" (CrAM) loss, defined as:

$$L^{\text{CrAM}}(\boldsymbol{\theta}) = \max_{\|\boldsymbol{\delta}\| \leq \rho} L(C(\boldsymbol{\theta} + \boldsymbol{\delta})), \tag{5.1}$$

where $\boldsymbol{\theta}$ is the vector of model parameters, L is the regular cross-entropy loss of the model, computed over the training set, and $\boldsymbol{\delta}$ is a norm-bounded perturbation. Unless otherwise stated, we employ the ℓ_2-norm throughout the rest of the chapter.

We approximate $\max_{\boldsymbol{\delta}} L(C(\boldsymbol{\theta} + \boldsymbol{\delta}))$ by taking a gradient ascent step in the direction of the current update, followed by a projection using the compression operator. This is inspired by the iterative hard thresholding (IHT) algorithm used for optimizing functions under sparse constraints [BD08, Fou11, Fou12]. To obtain the gradient with respect to the parameters, we employ a straight-through estimator, by using instead the gradient under the perturbation.

This gives us the following update for minimizing the CrAM loss:

$$\widetilde{\boldsymbol{\theta}}_t = C(\boldsymbol{\theta}_t + \rho \cdot \nabla L(\boldsymbol{\theta}_t)) \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\widetilde{\boldsymbol{\theta}}_t)\,. \tag{5.2}$$

In the update above we did not enforce the bound on the norm of the perturbation, as this has been shown to not be important for the original SAM algorithm [AF22], and in practice we also did not observe substantial differences compared to bounding the perturbation.

We note that solely optimizing the CrAM loss cannot offer guarantees for the performance of the dense model. Alongside improving robustness to compression, maintaining the quality of the dense model is one of the prerequisites of our method; therefore, we propose to also explicitly optimize for the performance of the dense model. Specifically, we optimize instead the following composite *CrAM+ loss function*:

$$L^{\text{CrAM}^+}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + L^{\text{CrAM}}(\boldsymbol{\theta})\,. \tag{5.3}$$

This can be achieved with a simple modification to the CrAM update, at no extra cost, by simply adding the gradient $\nabla L(\boldsymbol{\theta}_t)$, before the next update of the parameters $\boldsymbol{\theta}_{t+1}$. For

Algorithm 2 Compression-Aware Minimization (CrAM)

Require: Compression methods $\mathcal{C} = \{C_1, C_2, \ldots, C_M\}$, training data S, training iterations T, learning rate η, perturbation step size ρ

Initialize the weights $\boldsymbol{\theta}_0$
while $t \leq T$ **do**
  Sample batch $x \in S$
  Compute mini-batch loss $L(\boldsymbol{\theta}_t; x)$ and gradient $\boldsymbol{g}_t = \nabla L(\boldsymbol{\theta}_t; x)$
  Uniformly choose a compression method $C \in \mathcal{C}$
  Get perturbed weights $\widetilde{\boldsymbol{\theta}}_t = C(\boldsymbol{\theta}_t + \rho \boldsymbol{g}_t)$
  if $C =$ Top-K **then**
    Let M_t be the linear projection operator onto the support of the largest K coordinates of $|\boldsymbol{\theta}_t + \rho \boldsymbol{g}_t|$, such that $\widetilde{\boldsymbol{\theta}}_t = M_t(\boldsymbol{\theta}_t + \rho \boldsymbol{g}_t)$
    $\widetilde{\boldsymbol{g}}_t = M_t \nabla L(\widetilde{\boldsymbol{\theta}}_t; x)$
  else
    $\widetilde{\boldsymbol{g}}_t = \nabla L(\widetilde{\boldsymbol{\theta}}_t; x)$
  end if
  if use CrAM$^+$ **then**
    $\widetilde{\boldsymbol{g}}_t \leftarrow \widetilde{\boldsymbol{g}}_t + \boldsymbol{g}_t$
  end if
  Update the weights using a gradient descent step: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \widetilde{\boldsymbol{g}}_t$
end while
return $\boldsymbol{\theta}_T$

$\widetilde{\boldsymbol{\theta}}_t = C(\boldsymbol{\theta}_t + \rho \nabla L(\boldsymbol{\theta}_t))$, the CrAM$^+$ update is the following:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \left(\nabla L(\widetilde{\boldsymbol{\theta}}_t) + \nabla L(\boldsymbol{\theta}_t)\right). \tag{5.4}$$

We note that we can add different regularization terms to the objective in Equation 5.3; for example, in our experiments we use weight decay, as it is standard for training image classification models.

Overall, we observed in our experiments that using CrAM$^+$ resulted in an important improvement in dense model accuracy, relative to CrAM, without negatively impacting the accuracy of the resulting sparse models after one-shot pruning. Therefore, we propose using CrAM$^+$ as the main method in our experiments.

5.3.2 Theoretical Justification of the CrAM Update

To derive the CrAM update, and justify the choices made in designing our training method, we start from the optimization objective defined in Equation 5.1. As our goal is to minimize the CrAM loss L^{CrAM}, we use gradient descent. Using this loss complicates our objective, as it now includes an inner maximization problem, together with a potentially problematic compression operator. However, under mild assumptions we can efficiently estimate an approximate gradient which gives a descent direction.

To do so we rely on a well-known theorem from robust optimization [Dan12], which allows one to obtain descent directions for min-max objectives under a broad range of assumptions. Using Danskin's theorem (Theorem C.1.1 from Appendix C.1.1) we obtain that by computing the maximizer of the inner problem

$$\boldsymbol{\delta}^* = \arg\max_{\|\boldsymbol{\delta}\| \leq \rho} L(C(\boldsymbol{\theta} + \boldsymbol{\delta})), \tag{5.5}$$

and letting $\boldsymbol{\phi} = \boldsymbol{\theta} + \boldsymbol{\delta}^*$, which is compressed to the extrapolated iterate $\widetilde{\boldsymbol{\theta}} = C(\boldsymbol{\phi})$, we obtain a descent direction $-\nabla L(C(\boldsymbol{\phi}))$. Implementing this approach faces two difficulties – first, to compute the gradient we must back-propagate through the composition of functions $L(C(\cdot))$, which may cause trouble since C is not necessarily differentiable; second, and more importantly, it is unclear how to solve the inner maximization problem.

To address the first issue, we may choose to use a straight-through gradient estimator [BLC13], which permits us to only backpropagate through L, and use $\nabla L(\widetilde{\boldsymbol{\theta}})$ instead of the true gradient. To increase precision, in the case where compression is performed via Top-K we interpret C as a "mask" operator M which zeroes out a subset of coordinates dependent on $\boldsymbol{\phi}$. Since except for articulation points, M_t is constant and does not change as the argument varies, we approximate $\nabla L(C(\boldsymbol{\phi})) \approx M \nabla L(M\boldsymbol{\phi}) = M \nabla L(\widetilde{\boldsymbol{\theta}})$.

To address the second issue, rather than exactly maximizing the inner problem, we instead seek a good enough maximizer using a standard iterative method. For this, we choose projected gradient ascent, which provides theoretical guarantees, even when the projection is performed onto non-convex domains [PIVA21]. For instance, if the compression operator is magnitude pruning, this becomes the iterative hard thresholding (IHT) method, frequently employed in the sparse recovery literature [BD08]. Thus, to reach a good iterate within this specific domain, in practice we perform a single step of (projected) gradient ascent, which matches the IHT iteration:

$$\widetilde{\boldsymbol{\theta}}_t = C\left(\boldsymbol{\theta}_t + \rho \cdot \nabla L(\boldsymbol{\theta}_t)\right) . \tag{5.6}$$

We note that while the discussion above focused on theoretically justifying the CrAM update, it can easily be extended to CrAM$^+$ as well. First, we remind that the CrAM$^+$ loss is simply the sum between the loss of the dense model and the CrAM loss. Therefore, since the gradient naturally gives a descent direction for standard loss function, together with our previous result on the CrAM loss, we obtain a descent direction for the CrAM$^+$ loss.

In Appendix C.1.1, we provide a full re-derivation of the CrAM update in Equation 5.2 under fairly reasonable assumptions on the objective function, along with a detailed discussion on the necessity of these assumptions. As a side result, we also obtain a simple re-derivation of the SAM update.

5.3.3 Implementation Details and Extensions

Multiple compression types. CrAM can be used to train models that are robust to multiple types of compression operators. This can be enforced by choosing between multiple compression projections at each CrAM optimization step. Examples include pruning using different sparsity levels or quantizing at different precisions. We illustrate the general CrAM algorithm which handles multiple compression operators, and includes the explicit optimization of the dense model, in Algorithm 2. In our experiments, we found that applying the CrAM$^+$ update with a different randomly chosen compression at each optimization step typically achieves a good trade-off between a high dense model accuracy and robustness to multiple one-shot compression schemes post-training. When optimizing for robustness against sparse perturbations, we use the Top-K operator at each step, and choose the sparsity level uniformly at random among a set of predefined values.

Addressing the computational overhead of CrAM. Similar to the original SAM update, CrAM requires twice as many forward-backward passes, compared to a regular training cycle. In the case of TopK-CrAM, we can reduce this overhead, by making use of the sparsity in

the intermediate updates. Furthermore, we found that using only the gradients from the support of $\widetilde{\boldsymbol{\theta}}_t$ in $\nabla L(\widetilde{\boldsymbol{\theta}}_t)$ improves both the resulting dense model obtained with TopK-CrAM, as well as its robustness to one-shot pruning. This observation is motivated by the fact that the straight-through [BLC13] gradient estimator using the identity function is often times suboptimal [YLZ+19], and better straight-through estimators can be defined using different functions. As seen in Section 5.3.2, we can assume, via Danskin's theorem [Dan12], that we can obtain descent directions for $L^{\text{CrAM}}(\boldsymbol{\theta}_t)$ by evaluating $\nabla L(C(\boldsymbol{\phi}_t))$, where $\boldsymbol{\phi}_t$ is the extrapolated point $\boldsymbol{\phi}_t = \boldsymbol{\theta}_t + \rho \nabla L(\boldsymbol{\theta}_t)$. To evaluate the gradient, we may use a straight-through estimator. For Top-K, C is as an operator M_t which zeroes out a subset of coordinates dependent on $\boldsymbol{\phi}_t$. Provided that M_t is constant and does not change as the argument varies, we can approximate $\nabla L(C(\boldsymbol{\phi}_t)) \sim M_t \cdot \nabla L(M_t \boldsymbol{\phi}_t)$. As both the iterate and gradient estimator are sparse, this implies a theoretical speed-up. Furthermore, we note that at each CrAM update, we recompute the masks associated with the Top-K operator applied to the model's parameters. We show in Section 5.4.1 that it is possible to still obtain competitive results with CrAM when the masks are kept fixed for most of the updates, and only updated periodically; having less frequent masks updates reduces the time required for a CrAM update, since the Top-K operator is skipped for most of the steps, and also is aligned with our intuition that there are only small differences between consecutive masks.

Alternative Updates. We note that alternative compression-aware updates can be derived. For example, by following similar derivations to those developed for SAM [FKMN21], we get $\widetilde{\boldsymbol{\theta}}_t = C\left(\boldsymbol{\theta}_t + \rho \frac{\nabla L(C(\boldsymbol{\theta}_t))}{\|\nabla L(C(\boldsymbol{\theta}_t))\|}\right)$. We call this update *Compressed-SAM (C-SAM)*. We observed that training with C-SAM can also result in models that are robust to one-shot pruning, but typically the accuracy of the resulting dense models is lower, compared to training with CrAM. Moreover, training with C-SAM cannot offer guarantees for the performance of the dense model. While with CrAM we can optimize the dense model loss for free (i.e. using CrAM+), with C-SAM optimizing for the dense model explicitly would require a third forward-backward pass at each training step. Additionally, we examine the importance of the extra gradient step in CrAM, by comparing against simply applying the Top-K operator to the parameters. We provide an ablation study in Section 5.6.2.

Statistics Correction. It is well-known [HCI+21, FA22a] that pruning weights in a single step at high sparsity levels can have a large negative effect on normalization layers, due to a mismatch between layer statistics, e.g. the running mean and variance of BatchNorm layers, computed during training, and those of the pruned model. To correct for this, following prior work, we keep a subset of randomly chosen 1000 training samples (e.g. for ImageNet one sample per class), to which we apply standard training augmentations, and which are used post-pruning for resetting the Batch Norm statistics of the sparse model. We note that this procedure, which we refer to as BatchNorm Tuning (BNT) is very inexpensive, and does not finetune any other parameters of the model. Furthermore, during CrAM training on image classification models we only track the BatchNorm statistics on the dense model, before applying the compression perturbation. In the case of BERT models, we do not apply any statistics corrections.

5.4 Image Classification Experiments

Our experimental validation mainly focuses on sparsity, obtained by applying the Top-K operator, in the context of CrAM (i.e. TopK-CrAM). We denote the CrAM runs by the sparsity

level used during training. For example, "CrAM-k50" indicates that the Top-K operator with k=50% was used at each step, while "CrAM-Multi" indicates that the sparsity level is chosen uniformly at random, at each step, from a set of given values (e.g. CrAM-k$\{50, 70, 90\}$). For image classification experiments, all one-shot pruning results are presented after Batch Norm tuning (BNT) on a subset of 1000 training samples, i.e. 100 inference steps on batches of size 128, using standard random augmentations.

5.4.1 ImageNet Experiments

General Setup. We use a standard setup for training our ImageNet/ResNet50 models, similar to [FKMN21]. Namely, we use standard data augmentation, and we train the models using SGD for 100 epochs, with batch size 512, momentum 0.9, and weight decay 0.0001. The learning rate is linearly increased for the first 5 epochs until it reaches a maximum value of 0.2, after which it is decreased at each epoch, using a cosine scheduler. To determine the value of the hyperparameter ρ, we search over a small grid, by training 90% of ImageNet using CrAM-k50, and using the remaining 10% of the dataset for validation. We have found $\rho = 0.05$ to give good results for CrAM-k50, in terms of validation accuracy of the dense model, and we have kept this value for all our other CrAM experiments. For SAM, we also use $\rho = 0.05$, which is the standard value recommended by [FKMN21]. To match the number of backpropagation steps of CrAM, we additionally train the dense baseline for twice as many epochs. With these training hyperparameters, the standard dense baseline reaches 76.9% validation accuracy, which improves to 77.2% with extended training, while the SAM model reaches 77.4% accuracy. As stated, after one-shot pruning, we perform BNT on a subset of 1000 training samples (e.g. one per class), with standard augmentations. We show in Appendix C.2.1 that the accuracy after BNT is extremely stable, with respect to the choice of calibration set.

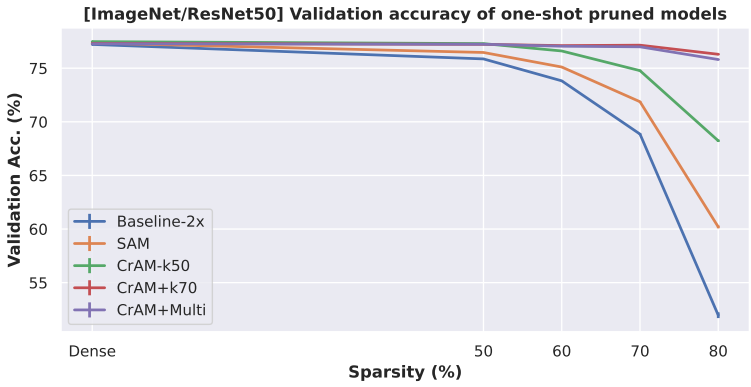


Figure 5.1: One shot pruning results, after BNT. Results are averaged across 10 independent BNT trials using randomly chosen calibration sets of 1000 samples.

Model	Sparsity 80%	90%
CrAM$^+$-Multi	75.8	74.7
WoodFisher	76.7	75.3
AC/DC	76.2	75.2
STR	76.1	74.3
DPF	75.1	74.6

Table 5.1: One shot pruned (+BNT) CrAM$^+$-Multi models vs. existing pruning methods.

Results for one-shot pruning. We validate the robustness to post-training compression of models trained through different versions of CrAM, by testing their accuracy after one-shot pruning, at different sparsity levels. We train models using CrAM-k50, CrAM$^+$-k70 and CrAM$^+$-Multi, where for the latter we choose uniformly at random, at each step, among 50%, 70% or 90% global sparsity levels. Using CrAM$^+$ is crucial for preserving the dense model accuracy, when using higher sparsities during training ($\geq$ 70%); however, when training with

low sparsities (e.g. CrAM-k50), the resulting dense model is slightly better than the baseline, with a 77.5% validation accuracy. For both CrAM$^+$-k70 and CrAM$^+$-Multi we use sparse gradients for the Top-K model perturbation, as described in Section 5.3.2. This improved substantially the accuracy after one-shot pruning, as well as the resulting dense model. Namely, the CrAM$^+$-Multi dense model had 77.3% validation accuracy, while the CrAM$^+$-k70 dense model had 77.2% accuracy. An additional benefit of using sparse gradients is the potential for training-time speed-up compared to, for example, using dense gradients or training with SAM, with the right framework support. We include an ablation study on the effects of sparse gradients in Section 5.6.1.

The results from Figure 5.1 show that CrAM models are substantially more robust to one-shot pruning, compared to standard SGD or SAM training. CrAM models do not lose accuracy at lower sparsity (e.g. at 50% for all or 70% for CrAM$^+$-k70 and CrAM$^+$-Multi). Moreover, as shown in Table 5.1, the results at higher sparsity (80% and 90%) levels are competitive with those obtained by gradual pruning, such as WoodFisher [SA20] or methods that prune during training– STR [KRS$^+$20], DPF [LSB$^+$20], or AC/DC [PIVA21]. We emphasize that CrAM requires a single round of training (albeit with twice as many forward-backward passes, compared to regular SGD), while standard pruning methods require training separately for each target sparsity, sometimes from a pretrained model (e.g. WoodFisher).

In addition to global magnitude, we show that CrAM models can be trained to be robust to different sparsity distributions, such as uniform. We train CrAM$^+$-Multi models for ImageNet/ResNet50 under the same setup as for global magnitude, but applying instead the Top-K operator at uniform sparsity across all prunable parameters (i.e. excluding BatchNorm and biases); additionally, we keep the first and last layers dense, as these have been shown to have an important impact on accuracy, without any effect on inference speed-up. The resulting dense model achieves 77.1% accuracy, while one-shot uniform pruning at 80% and 90% sparsities gives, after BNT, 75.6% and 75.1% accuracy, respectively. Moreover, this model is also robust to one-shot pruning using global magnitude (e.g. 75.5% accuracy at 80% sparsity). Conversely, CrAM$^+$-Multi trained with global magnitude is robust to one-shot pruning using uniform magnitude (e.g. 77.0% and 75.9% accuracy at 70% and 80% sparsity, respectively). This suggests that CrAM-trained models can be robust to one-shot pruning using sparsity distributions different from the ones used during training.

Results for N:M sparsity patterns. Additionally, we show that CrAM can be successfully used with semi-structured N:M sparsity patterns, where out of each block of M weights, N are sparse, as discussed in Section 2.2.3 from Chapter 2. We train CrAM$^+$ models with the N:M pattern, by randomly choosing at each optimization step between the 2:4 or 4:8 projections; this model will be referred to as "CrAM$^+$-N:M". Similar to the previous experiments, we also use sparse gradients for the pruned model perturbation and have found this to have a positive impact on the one-shot pruned models. We provide more details on the effect of sparse intermediate gradients in Section 5.6.1. In Table 5.2 we show the one-shot pruning results (after BNT with 1000 samples). For completeness, we also include the accuracies when pruning the dense, SAM and CrAM-k50 models, using the N:M patterns. Note that CrAM$^+$-N:M models do not lose accuracy when they are pruned one-shot using the 2:4 and 4:8 patterns, which is competitve with state-of-the-art methods for training N:M sparse models, such as SR-STE [ZMZ$^+$21]. However, we point out that SR-STE requires retraining a separate model from scratch, for each sparsity profile. Additionally, we observe that the CrAM$^+$-N:M model is robust to one-shot pruning using also unstructured patterns, at moderate sparsity levels; for example, for global and uniform pruning, models trained with CrAM$^+$-N:M do not

lose any accuracy when pruned one-shot to 50% sparsity (77.3% accuracy), and have only a minor accuracy loss, compared to the dense baseline, when pruned to 70% sparsity (76.5% accuracy).

Model	Dense	Sparsity Pattern	
		2:4	4:8
Dense	77.2	66.5	69.6
SAM	77.4	69.6	72.0
CrAM-k50	**77.5**	72.2	73.4
CrAM$^+$-N:M	77.3	**77.0**	77.2
SR-STE	-	**77.0**	**77.4**

Table 5.2: (ImageNet/ResNet50) Validation accuracy (%) after one-shot pruning (+BNT) using semi-structured 2:4 and 4:8 patterns.

Model	ResNet18		ResNet50	
	Dense	4 Bits	Dense	4 Bits
Baseline	69.8	66.2	76.1	74.1
SAM	**70.5**	67.1	**76.9**	74.8
CrAM$^+$-4Bits	70.1	**69.3**	76.7	**75.8**

Table 5.3: (ImageNet) Validation accuracy (%) for the dense models, and after symmetric per-channel 4 bits quantization. All quantization results are after BNT.

Finetuning with CrAM. To reduce the computational overhead of CrAM-training, we investigate whether a dense model's robustness to pruning can be improved with only a short finetuning using CrAM. This approach is inspired by [AF22], which shows encouraging evidence that similar benefits to fully training with SAM can be obtained when the latter is instead used only in the final training phase. We finetune pretrained ImageNet ResNet18 and ResNet50 models, from the Torchvision library, using CrAM$^+$-k70 and CrAM$^+$-Multi, both with sparse gradients for the pruned perturbation. We perform finetuning for 10 epochs, starting from a learning rate of 0.005, which is decayed using a cosine learning rate scheduler, at each epoch. For CrAM$^+$-Multi we randomly select at each step a sparsity level in the range 50%-90%. For comparison, we also finetuned using SGD with momentum or using SAM, under the same hyperparameters. We report in Tables 5.4 and 5.5 the validation accuracy for the dense models, and after one-shot pruning at 50%-80% sparsity levels. Finetuning with CrAM$^+$ preserves or outperforms the baseline accuracy, and results in good sparse models after one-shot pruning, at moderate sparsity levels (up to 70%). Results improve with longer finetuning: after 20 epochs, the CrAM$^+$-Multi model can be pruned one-shot to 70% sparsity, with $\leq 1\%$ drop in accuracy, compared to the baseline.

Model	Dense	Sparsity			
		50%	60%	70%	80%
Baseline	69.8	68.4	66.6	62.4	50.4
Dense	70.4	68.9	67.0	62.3	50.1
SAM	70.5	69.2	67.4	63.4	52.2
CrAM$^+$-k70	70.3	69.5	68.8	**69.0**	65.0
CrAM$^+$-Multi	70.4	69.7	69.2	68.3	66.7
CrAM$^+$-Multi-20	**70.6**	**69.9**	**69.6**	**69.0**	**67.6**

Table 5.4: (ImageNet/ResNet18) Accuracy after finetuning for dense models, and after one shot pruning

Model	Dense	Sparsity			
		50%	60%	70%	80%
Baseline	76.1	75.1	73.4	69.5	54.3
Dense	76.8	75.4	73.6	69.0	53.1
SAM	**76.9**	75.8	74.3	70.5	57.8
CrAM$^+$-k70	76.8	75.9	75.5	75.4	72.0
CrAM$^+$-Multi	76.7	75.9	75.6	75.0	73.5
CrAM$^+$-Multi-20	76.8	**76.1**	**75.7**	**75.5**	**74.4**

Table 5.5: (ImageNet/ResNet50) Accuracy after finetuning for the dense models, and after one shot pruning

CrAM with Infrequent Masks Updates. As previously mentioned, when training image models with CrAM$^+$, using sparse gradients of the intermediate model perturbation substantially improved both the accuracy of the dense final model, as well as after one-shot pruning. One hypothesis that would enable this approximation of the gradient would be that the masks of

the compression perturbation change very little during training. We test the validity of this hypothesis by training a version of CrAM$^+$ that only does infrequent mask updates. Namely, the masks obtained from the Top-K compression operator are kept fixed for a number of consecutive τ training iterations. In the case of CrAM-Multi, the masks for each sparsity level are changed each after τ iterations with the corresponding sparsity target. We note that infrequent mask updates improve the practical training time of CrAM, as the Top-K operator can be skipped for most iterations. This brings the cost of an average CrAM$^+$ iteration to the same level as a SAM iteration, though we note that in theory, with specialized hardware, the computational cost of CrAM can be further decreased due to the presence of sparse operators. We experiment using the same setup for training ImageNet on ResNet50 with CrAM$^+$-Multi, with global magnitude pruning at sparsity $k \in \{50\%, 70\%, 90\%\}$, and perform two separate runs, by varying the mask update frequency $\tau \in \{20, 100\}$ iterations. As before, we also use BNT after pruning, on a subset of 1000 training sample, consisting of one example per class. The results presented in Table 5.6 show that using infrequent mask updates only has a small impact on the initial results; namely, we observe that the results after one-shot pruning are slightly worse compared to the default ones, particularly at higher sparsity. In particular, the results for 80% sparsity have decreased the most, a level which was not explicitly used during training. We also repeated the same experiment in the CrAM-finetuning setup for ResNet50, with $\tau = 100$ iterations, and observed that results after one-shot pruning in fact improved slightly. This is particularly encouraging, as using CrAM only for finetuning is a more attractive use-case, due to the reduced computational costs.

Frequency	Sparsity (%)				
τ	0	50	70	80	90
1	77.3	77.2	77.0	75.8	74.7
20	77.4	77.4	77.2	75.5	74.8
100	77.3	77.3	76.9	75.3	74.5

Table 5.6: (ImageNet/ResNet50) Validation accuracy (%) for the dense and sparse CrAM$^+$-Multi models trained with sparse perturbed gradients, using infrequent mask updates of the global Top-K operator.

Quantization. While the previous experiments showed that CrAM can be successfully used to train models robust to one-shot pruning through the Top-K operator, we demonstrate that CrAM can also be used with other types of compression. Namely, we show encouraging evidence that CrAM can be adapted to quantization. Specifically, we use CrAM$^+$ where the compression operator C is the symmetric per-channel weight quantization to 4 bits (round to nearest integer), and finetune pretrained Torchvision ResNet18 and ResNet50 ImageNet models, for 10 epochs, using the same hyperparameters as in the previous experiments for sparsity. The results in Table 5.3 show that CrAM-finetuned models are more robust to symmetric per-channel 4 bits quantization, compared to models finetuned with SAM or with the dense baseline.

5.4.2 Using CrAM Models for Sparse Transfer

As previously stated, one of the main goals we want to achieve through CrAM is to train dense models that are easily compressible. One natural and practical use case for this property would be sparse transfer. Namely, an user would like to deploy a pre-trained model and adapt it to a specialized task, through finetuning. As described in Chapter 4, the devices used for

finetuning a model on specialized tasks often times pose computational constraints, and so the user might consider using a sparse version of the model; this would allow them to benefit from potential memory and computational savings arising from the sparsity of the network. In many cases, the sparse models are not readily available, and obtaining them would require large computational resources, in addition to access to the full upstream dataset used for training them. Therefore, it would be very useful if the large model could be pruned one-shot, without any retraining, nor access to the upstream dataset, in such a way that performing sparse transfer would still be competitive with the dense transfer, or with sparse transfer from a model pruned using traditional methods (e.g. GMP).

With this motivation in mind, we present an application which shows that sparse CrAM models can be successfully used for sparse transfer learning. For this purpose, we showcase the CrAM$^+$-Multi model from the previous section, trained on ImageNet and the 80% and 90% sparse models derived after one-shot pruning. To set a baseline for our comparison, we additionally consider the sparse models, without any retraining, obtained from pruning one-shot to 80% and 90% sparsity the dense ImageNet baselines (trained for 100 or 200 epochs), as well as the SAM model and the dense CrAM-k50 model trained on ImageNet. We further note that for the CrAM models, as well as the baselines, we consider the one-shot pruned models before BNT, since Batch Norm statistics are anyway corrected during finetuning, to reflect the change in data distribution.

We are interested in being competitive, in terms of sparse transfer performance, with sparse models that were a-priori trained on ImageNet, using existing pruning methods from the literature. For this purpose, we employ the setup from Chapter 4, and we choose for comparison sparse regularization methods (AC/DC [PIVA21] and STR [KRS$^+$20]), progressive sparsification methods (WoodFisher [SA20]) and lottery-tickets-based methods (LTH-T [CFC$^+$21]), each trained for 80% and 90% sparsity levels. In terms of transfer tasks and training hyperparameters, we use the same setup for full transfer from Chapter 4. We also use the same average relative increase in error metric, which aggregates the results for each method, at each sparsity level, across all twelve tasks. Compared to the previous chapter and [IPKA22], however, we replaced the dense baseline with the one trained under the same hyperparameters as CrAM; subsequently, the aggregated metrics for the pruning methods from Chapter 4 have been updated to be compatible with the current dense baseline.

The results in Figure 5.2 show that one-shot pruned CrAM models transfer well. In fact, both CrAM-k50 and CrAM$^+$-Multi models transfer better than LTH-T at 80% sparsity, although pruning is performed in one-shot. Also, CrAM$^+$-Multi at 90% sparsity has a similar transfer performance to AC/DC models, and gives better results compared to the other pruning methods used for comparison (LTH-T or STR), with the exception of the second-order WoodFisher method, which is the best performing method across both 80% and 90% sparsity, as also previously discussed in Chapter 4. Compared to the standard pruning methods used for comparison, CrAM has the added advantage that it produces an accurate dense model, and both 80% and 90% sparse models from a single ImageNet run. Therefore, these results show that CrAM models, and in particular CrAM$^+$-Multi, show a level of flexibility which allows a user to easily choose between multiple sparsity levels, depending on the computational resources available, at no extra cost. Moreover, the dense CrAM$^+$-Multi model has a very similar transfer performance to the baseline, with a less than 1% average relative increase in error, while dense CrAM-k50 or SAM, as well as the baseline trained for twice more iterations result in a minor improvement in transfer accuracy of around 2%, compared to the baseline.

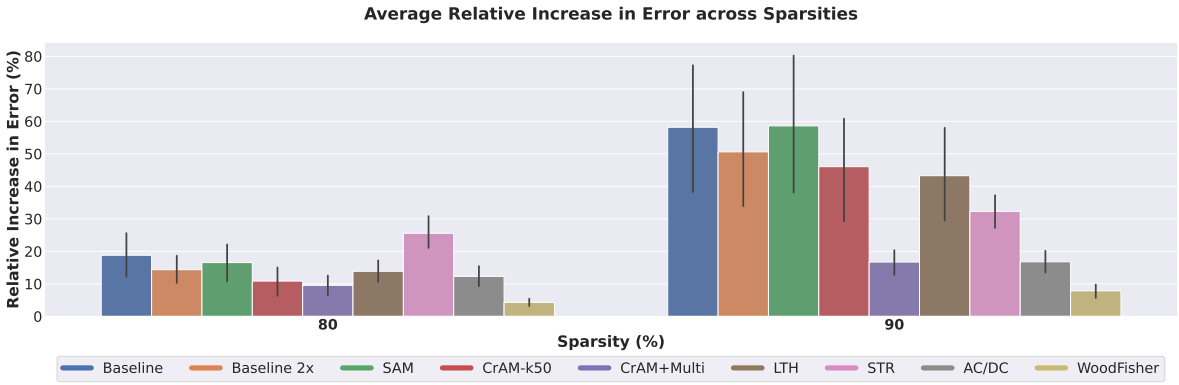


Figure 5.2: Average relative increase in error, relative to dense, on 12 tasks, between models pruned one-shot, or obtained from pruning methods, at different sparsities. Lower is better. All models were pretrained on ImageNet/ResNet50. For better visibility, error bars indicate 70% confidence intervals.

5.4.3 Detailed Comparisons with Other Methods

We now perform a detailed comparison between CrAM and gradual pruning methods, or similar methods that train a prunable dense model. We choose to focus on the CIFAR-10 dataset [KH+09] for the purpose of this comparison, since most of the other methods present experiments and are tuned on this particular dataset. Specifically, we compare CrAM with existing state-of-the-art gradual pruning methods [LSB+20] on ResNet20 [HZRS16], or with similar methods for training prunable networks [MLC+22, ZSP22] on VGG-16 [SZ14] and ResNet18.

Hyperparameters and general setup. For this set of experiments, we train ResNet20 models for 200 epochs, using SGD with momentum and weight decay, and a cosine learning rate scheduler, with a learning rate linear warm-up of 5 epochs. Additionally, we train the baseline model for twice as many epochs, to match the number of backpropagation steps of SAM and CrAM. To determine the value of the hyperparameter ρ, we performed a grid search over values in the range $0.01 - 0.2$, using a $90\% - 10\%$ train-validation split and found 0.1 and 0.15 to be the best values for SAM and CrAM+-Multi, respectively (i.e. achieving highest validation accuracy). After finding the best value of ρ for each model configuration, we retrained using the entire training set, and starting from 3 different random seeds, and report the final accuracy after 200 epochs of training. We follow a very similar training recipe and hyperparameter search for ResNet18 and VGG experiments, but train instead for 180 epochs, similarly to [MLC+22].

Comparison with Gradual Methods. In addition to the ImageNet results, we further demonstrate on CIFAR-10 the ability of CrAM to produce sparse models that are competitive with state-of-the-art gradual pruning methods. Namely, we showcase CrAM+-Multi, trained on ResNet20 using sparse intermediate gradients, where at each optimization step we select the sparsity level uniformly at random among values in the set $\{50\%, 70\%, 80\%, 90\%, 95\%\}$. The results in Table 5.7 show that the *dense* model obtained by training with CrAM+-Multi is highly accurate, achieving an accuracy close to the regular SGD baseline (93%), but below the model trained with SAM (93.5% accuracy). Moreover, CrAM+-Multi is very robust to one-shot pruning, even at high sparsities (e.g. 90% or 95%). We note that our results for one-shot pruning (+BNT) are usually competitive with those obtained by methods that train sparse models separately from scratch, for each sparsity target. For example, in Table 5.7 we compare

one-shot pruned models obtained from CrAM$^+$-Multi at various sparsity levels, with those obtained by training separately sparse models through the state-of-the-art dynamic model pruning with feedback (DPF) method [LSB$^+$20]. Remarkably, we obtain better accuracy after one-shot pruning than DPF models at all sparsity levels considered, even at 95% sparsity, where CrAM$^+$-Multi substantially outperforms DPF, which trains a separate model for each sparsity level. Additionally, we show a similar comparison on the larger VGG-16 architecture [SZ14]. Here, we train CrAM$^+$ with 95% sparsity at each step, using intermediate sparse gradients. As it can be observed from Table 5.7, CrAM$^+$-k95 produces models that do not lose accuracy (after BNT) up to 95% sparsity. In comparison, training a dense model using SGD under the same hyperparameter setup produces a model with 93.9% test accuracy. Moreover, when pruning CrAM$^+$-k95 at 95% sparsity we obtain a similar test accuracy to the one reported by DPF [LSB$^+$20], which explicitly trains a 95% sparse model.

Architecture	Model	Dense	Sparsity 50%	70%	80%	90%	95%
ResNet20	CrAM$^+$-Multi	92.9 ± 0.	92.8 ± 0.1	**92.7 ± 0.2**	**92.6 ± 0.2**	**91.2 ± 0.1**	**89.2 ± 0.1**
	DPF	N/A	N/A	92.4 ± 0.1	92.2 ± 0.2	90.9 ± 0.1	88.0 ± 0.3
VGG-16	CrAM$^+$-k95	**94.2 ± 0.1**	**94.2 ± 0.1**	**94.2 ± 0.1**	**94.1 ± 0.1**	**94.0 ± 0.1**	**94.1 ± 0.1**
	SFW	N/A	93.1	93.1	93.1	93.1	92.0
	DPF	N/A	N/A	N/A	N/A	N/A	93.9 ± 0.2

Table 5.7: (CIFAR-10) Test accuracy (%) for CrAM after one shot pruning (+BNT). CrAM$^+$-Multi is competitive with or outperforms state-of-the-art pruning method DPF, up to 95% sparsity on ResNet20 and VGG-16. DPF requires retraining for each target sparsity. CrAM$^+$ outperforms similar method SFW [MLC$^+$22]

Comparison with One-shot Pruning Methods. In addition to being competitive with gradual pruning methods, we show that CrAM produces better sparse models than similar methods for training prunable models. Namely, we compare against the recent works [MLC$^+$22, ZSP22], which propose methods based on Stochastic Frank-Wolfe (SFW) [RSPS16]; SFW is a "gradient-free" optimization algorithm, which encourages the parameters to lie in the convex hull spanned by sparse vectors, with directions given by the gradients. We compare CrAM against SFW on CIFAR-10, using ResNet18 [HZRS16] and VGG-16 models, which is the same experimental setup employed in [MLC$^+$22] or [ZSP22]. Note that [MLC$^+$22, ZSP22] use the "ImageNet" variant of the ResNet18 model, which is substantially larger than the ResNet20 used for our previous experiments (11M vs. 0.3M parameters). As previously described, we train CrAM$^+$-k95 with sparse intermediate gradients, under the same hyperparameter setup on both ResNet18 and VGG-16, except for the ρ value determined separately by grid search, and we use BNT after pruning at each desired sparsity level.

On both ResNet18 and VGG-16, we obtain dense models that do not lose accuracy compared to the baseline: 94.2% (CrAM$^+$-k95) vs. 93.9% (dense) on VGG-16 and 95.7% (CrAM$^+$-k95) vs. 95.4% (dense) on ResNet18. Furthermore, on ResNet18 we maintain the model accuracy after pruning one-shot at 96% sparsity (95.4%, after BNT) and have a 1.3% drop at 98% sparsity (94.4% Top-1). These accuracies are higher than those from [MLC$^+$22] and [ZSP22], who obtain $\leq$ 93% accuracy at 95% sparsity. We show a numerical comparison for VGG-16 in Table 5.7: CrAM$^+$-k95 preserves model accuracy even at 95% sparsity, while SFW produces models that have lower accuracy even at higher density. We note that CrAM has higher training costs than SFW, but requires less hyper-parameter tuning, and leads to higher accuracies for the dense and sparse models. Moreover, similar to [ZSP22], our method uses

BNT, while [MLC+22] do not suggest that they are using it. However, the cost of BNT is minimal; even *without BNT*, our method preserves accuracy at up to 80% sparsity (please see Appendix C.2.2), leading to better results than [MLC+22, ZSP22].

5.5 Experiments on Language Modeling

In addition to the image classification results presented in Section 5.4, we show that CrAM can be successfully used with different applications, such as language models. We demonstrate that CrAM produces models that are more compressible and accurate than the ones obtained with standard optimizers such as Adam [KB15] and SAM [FKMN21]. Furthermore, sparse CrAM models are even competitive with gradual pruning methods; the latter usually require a higher computational budget, as they separately train models for each sparsity target.

General setup. For our experiments, we focus on a standard benchmark also used by other compression methods, namely the BERT-base [DCLT19] model finetuned on the span-based question-answering task SQuADv1.1 [RZLL16]. BERT-base [DCLT19] is a commonly used language model, which consists of 12 identical transformer layers [WDS+20], consisting of 110M parameters. Following the community standards (e.g. [SWR20, KCN+22]) we sparsify all weights of the encoder part, which consists of 85M parameters, and report sparsities relative to this number. For pruning, we make use of the Top-K operator at each step to impose uniform sparsity distribution over all layers. Our experiments are performed using open-source libraries, such as Transformers [WDS+20] and SparseML [KKG+20] and open source datasets obtained via [LVdMJ+21].

5.5.1 Robustness to one-shot pruning

We consider the setup in which the pretrained BERT-base model is finetuned for a small number of epochs on a downstream task. After finetuning the models using Adam, SAM, and several variants of CrAM, we finally test their robustness to one-shot magnitude pruning using uniform sparsity.

To identify the optimal set of hyper-parameters we run a grid search for each optimizer independently and pick the configuration with the best one-shot performance at 50% sparsity target. For a fair comparison, we further allow Adam to fine-tune for twice as many epochs as SAM and CrAM. The complete set of hyperparameters used for grid search, as well as the final configurations for each model, are provided in Appendix C.3.

We train different version of CrAM+ models, each using the Top-K operator at each step with uniform sparsity. Unlike the image classification experiments, we use dense gradients in the parameter update; in our experience, using sparse gradients with language models resulted in a decreased performance at lower sparsity levels, with a slight increase at higher sparsities. In the case of CrAM+-Multi, we select uniformly at random, at each optimization step, sparsity levels in the set $\{50\%, 60\%, 70\%, 80\%\}$.

The results presented in Table 5.8 suggest that CrAM models are more robust to one-shot pruning while still being able to match or even outperform the dense F1 scores obtained with other optimizers. Furthermore, we explore whether it is possible to reduce the computation complexity of CrAM, without sacrificing performance. For this purpose, we finetune models which use the CrAM updates only on a fraction of the total training updates; specifically, at each optimization step we choose with a certain probability between the CrAM or a standard

Adam update. In Table 5.9 we provide results for this experiment, in which at every separate run we use Adam with a certain probability, e.g. if $\mathrm{p(Adam)} = 0.8$ CrAM is used only on 20% of the optimization steps. The results show that even in this restricted setup, models become substantially more robust to pruning, compared to, for example, only using Adam at each step.

Model	Dense	Sparsity			
		50%	60%	70%	80%
Adam	88.7	80.0	32.5	9.6	8.1
SAM	88.5	81.0	33.4	10.1	7.3
CrAM$^+$-k50	**88.9**	**88.3**	84.6	25.3	8.3
CrAM$^+$-k60	88.7	88.1	87.8	75.7	10.2
CrAM$^+$-k70	88.8	87.8	87.0	**86.9**	33.9
CrAM$^+$-k80	88.4	86.9	85.5	84.9	**84.7**
CrAM$^+$-Multi	88.7	**88.3**	**88.1**	86.8	82.5

Table 5.8: (SQuADv1.1/BERT-base) Validation F1 score of models after fine-tuning with the corresponding optimizer and applying one-shot magnitude pruning.

Model	Dense	Sparsity			
		50%	60%	70%	80%
CrAM	88.7	**88.3**	**88.1**	**86.8**	82.5
$\mathrm{p(Adam)} = 0.1$	87.6	87.4	87.4	86.5	**84.0**
$\mathrm{p(Adam)} = 0.3$	87.5	87.5	87.2	86.5	83.6
$\mathrm{p(Adam)} = 0.5$	87.8	87.7	87.2	86.4	83.0
$\mathrm{p(Adam)} = 0.8$	87.0	87.1	86.8	85.2	79.1

Table 5.9: (SQuADv1.1/BERT-base) Validation F1 score of models optimized with CrAM$^+$-Multi where at each step with probability $\mathrm{p(Adam)}$ the standard Adam step is applied instead of the CrAM$^+$-Multi step.

Comparison with gradual pruning methods. We further investigate whether one-shot pruned CrAM models are competitive with models produced by gradual pruning methods, which progressively prune smaller fractions of weights and fine-tune the model for many epochs. We use the CrAM$^+$-Multi model from Table 5.8 and prune it in one-shot with the standard uniform magnitude pruner, but also using oBERT [KCN$^+$22], which is a state-of-the-art pruning method for BERT models. The oBERT method takes into account the second order information to decide which weights to prune, similar to other pruning methods developed for this purpose [SA20, FKA21]. In Table 5.10, we compare sparse CrAM models with different gradual pruning methods, such as: ℓ_0 regularization [LWK18], Magnitude [ZG17], Movement [SWR20], Soft-Movement [SWR20] and PLATON [ZZL$^+$22]. The results show that one-shot pruned CrAM$^+$-Multi models surpass state-of-the-art gradual pruning methods, such as PLATON [ZZL$^+$22] at low to moderate sparsity levels ($\leq 70\%$).

Model	Pruning	Sparsity			
		50%	60%	70%	80%
ℓ_0 regularization	gradual	84.6	83.9	82.8	81.9
Magnitude	gradual	87.0	86.7	86.5	84.8
Movement	gradual	83.0	82.8	81.9	82.0
Soft-Movement	gradual	85.8	N.A.	84.6	N.A.
PLATON	gradual	87.2	86.9	86.7	86.1
CrAM$^+$-Multi	one-shot magnitude	88.3	88.1	86.8	82.5
	one-shot oBERT	**88.7**	88.1	87.5	84.9
	one-shot oBERT + fine-tune	**88.7**	**88.4**	**88.1**	**87.4**

Table 5.10: (SQuADv1.1/BERT-base) Validation F1 score of the CrAM$^+$-Multi model after one-shot pruning with magnitude and oBERT pruners. We additionally fine-tune the one-shot oBERT-pruned model and compare it with gradual pruning methods.

However, one-shot pruning to high sparsity targets can severely impact the model's performance, as it can be observed for CrAM$^+$-Multi at 80% sparsity. Therefore, we also investigate whether a short fine-tuning period (for at most 2 epochs) using the sparse CrAM models, with fixed masks, can bridge the gap towards full accuracy recovery. Table 5.10 further contains these

additional results, which show that finetuned CrAM models substantially outperform gradual pruning methods at all sparsities considered ($\leq 80\%$). It is worth emphasizing that the competitive results obtained with two different one-shot pruners, magnitude and oBERT, suggest that CrAM models are indeed robust and compatible with pruning techniques different from the ones they have been trained with. Furthermore, we provide in Table 5.11 inference speed-up numbers for the sparse BERT models, showing that , showing that we obtain more than 2x speed-up, compared to the dense model, for CrAM models at 80% sparsity. We provide complete details regarding hyper-parameters used for oBERT pruning and sparse finetuning in Appendix C.3.

Sparsity	**4-cores/batch-size=1** Throughput (items/sec)	Speed-up	**16-cores/batch-size=128** Throughput (items/sec)	Speed-up
Dense	4.0	1.0x	14.2	1.0x
50%	4.5	1.1x	18.0	1.3x
60%	5.2	1.3x	21.8	1.5x
70%	6.3	1.6x	26.0	1.8x
80%	8.0	2.0x	31.9	2.3x

Table 5.11: (SQuADv1.1/BERT-base) Speed-ups of pruned BERT-base models relative to the dense model, benchmarked with the sparsity-aware inference engine DeepSparse (version 1.0.2) [KKG+20, Dee21] in two different scenarios on AMD EPYC 7702 64-Core Processor.

5.6 Ablation Studies for the CrAM update

In this section we investigate the impact that different modifications to CrAM can have on both the final dense model accuracy, as well as on the resulting one-shot pruned models. Furthermore, we study alternative updates to CrAM, for training prunable models, and we show that CrAM achieves the best trade-off between computational complexity and quality of both dense and sparse models.

5.6.1 Importance of Sparse Gradients

Model	Dense	Sparsity 50%	70%	80%	90%
CrAM-k70	75.7	76.3	76.3	73.4	53.2
CrAM+-k70	**77.3**	**77.3**	76.8	73.9	51.9
CrAM+-k70 (SG)	**77.3**	77.2	**77.2**	**76.3**	**62.1**
CrAM-Multi	75.2	75.2	75.2	74.5	73.3
CrAM+-Multi	76.4	76.4	76.1	74.9	73.1
CrAM+-Multi-SG	**77.3**	**77.2**	**77.0**	**75.8**	**74.8**

Table 5.12: (ImageNet/ResNet50) Dense and one-shot pruning (+BNT) results. CrAM+-SG improves the accuracy of the dense model, and its robustness to one-shot pruning.

Model	Dense	Sparsity Pattern 2:4	4:8
CrAM-N:M	75.2	76.0	76.2
CrAM+-N:M	77.1	76.1	76.6
CrAM+-N:M-SG	**77.3**	**77.0**	**77.2**

Table 5.13: (ImageNet/ResNet50) Dense and semi-structured one-shot pruning (+BNT) results. CrAM+-N:M-SG improves the robustness to N:M pruning, compared to CrAM+.

As previously discussed in Sections 5.4.1 and 5.4.3, we observed for image classification experiments an improvement in the robustness to post-training one-shot pruning, when using

sparse intermediate gradients. This modification corresponds to using a different straight-through estimator for the gradient $\nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}) = \nabla_{\boldsymbol{\theta}} L(C(\boldsymbol{\theta} + \rho \nabla L(\boldsymbol{\theta})))$, which instead of by-passing the Top-K operator in the gradient, and estimating $\nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}) \approx \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\widetilde{\boldsymbol{\theta}}}$, assumes that the masks M of $\widetilde{\boldsymbol{\theta}}$ are almost constant during training. This would allow the use of the approximation $\nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}) \approx M \odot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\widetilde{\boldsymbol{\theta}}}$. We refer to this approximation of CrAM using sparse intermediate gradients as CrAM-SG. More explanations, as well as theoretical support for this approach, are provided in Section 5.3.2 and Appendix C.1. We observed that training ImageNet/ResNet50 models using CrAM$^+$-SG led to an improvement in the robustness to one-shot pruning, particularly at higher sparsity levels and a similar trend was also observed for CIFAR-10/ResNet20 models.

Furthermore, we can experimentally confirm our intuition that the masks are fairly constant during training. For example, on CIFAR-10/ResNet20, trained with CrAM$^+$-k70, the difference between consecutive $\widetilde{\boldsymbol{\theta}}_t$ masks was lower than 0.6%. Interestingly, using sparse gradient under the same training setup, encouraged more diversity in the masks; namely, the difference between consecutive masks at later training stages increased to around 2%. We speculate this could be a potential reason for the improved robustness to pruning. Another aspect observed on CIFAR-10 experiments was that using sparse gradients tends to decrease the dense model accuracy, when CrAM is trained with lower sparsity levels; for example, the dense model for CrAM-k50 reached 93.4% accuracy, which decreased to 92.8% when using sparse gradients. For this reason, on ImageNet we only experimented with the dense version of CrAM-k50. Nonetheless, using sparse gradients improved the robustness to pruning in all cases. The effects of using sparse intermediate gradients are illustrated in Table 5.12, where we provide the results obtained from training CrAM$^+$ and CrAM$^+$-SG on ImageNet, for one-shot unstructured global magnitude pruning. For completeness, we also include the results for CrAM models, which show the improvement to the dense model resulted from using CrAM$^+$. We provide the same comparison for CrAM models trained with semi-structured N:M sparsity patterns, in Table 5.13.

We further note, however, that in the case of language models, using sparse intermediate gradients did not have a similar positive effect. From our experiments, both CrAM-SG and CrAM$^+$-SG decreased the accuracy at lower sparsity levels, but slightly improved the results at high sparsity. For these reasons, we only used intermediate sparse gradients for our image classification experiments.

5.6.2 Comparison Between CrAM and Alternative Updates

We further investigate the importance of individual components from CrAM, by comparing against other similar updates. For the purpose of this comparison, we focus on the CIFAR-10 dataset, using a ResNet20 model. For all methods considered, we use the same training hyperparameters or grid search procedure for finding ρ as the one described in Section 5.4.3; for the "multi" versions of the algorithms we consider sparsity values sampled uniformly at random in the range $30\% - 90\%$. We study two different types of updates related to CrAM: one is derived from the SAM [FKMN21] update, where we compose the standard cross-entropy loss function with the compression operator; the second update is derived by simply eliminating the extra-gradient step in the the original CrAM update. These modifications would inform us about the effects of using the gradient of the dense model for perturbing the parameters before applying the compression operator, and also would help us understand the importance of the extra gradient step for obtaining prunable models with CrAM.

Alternative Updates Derived from SAM. One alternative update to CrAM can be obtained by following closely the derivations for SAM [FKMN21]. For this, we assume $\|\boldsymbol{\delta}\| \leq \rho$, define $h(\boldsymbol{x}) := L(C(\boldsymbol{x}))$ (with C the Top-K operator), and the loss $\max_{\|\boldsymbol{\delta}\| \leq \rho} h(\boldsymbol{\theta} + \boldsymbol{\delta})$. We assume that for a small enough ρ, h is differentiable around $\boldsymbol{\theta}$, and we use a first-order Taylor approximation of $h(\boldsymbol{\theta} + \boldsymbol{\delta})$ around $\boldsymbol{\theta}$, together with the quadratic constraint for $\boldsymbol{\delta}$ to define our maximization problem. We obtain after differentiation and applying the inequality constraints, that $\boldsymbol{\delta} = \rho \cdot \frac{\nabla L(C(\boldsymbol{\theta}))}{\|\nabla L(C(\boldsymbol{\theta}))\|}$. This enables us to define the compressed-SAM (C-SAM) update as:

$$\text{C-SAM:} \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \nabla L \left(C \left(\boldsymbol{\theta}_t + \rho \cdot \frac{\nabla L(C(\boldsymbol{\theta}_t))}{\|\nabla L(C(\boldsymbol{\theta}_t))\|} \right) \right) . \tag{5.7}$$

To derive the Taylor approximation of $h(\boldsymbol{\theta} + \boldsymbol{\delta})$ around $\boldsymbol{\theta}$, we need to use a straight-through estimator for the gradient $\nabla_{\boldsymbol{\theta}} L(C(\boldsymbol{\theta}))$, due to the non-differentiability of C. Similar to our previous discussion about CrAM, we observed that C-SAM training benefits from using sparsified gradients; here, we use sparse gradients for both the model perturbation in the intermediate interpolation step, as well as in the final weight update. Namely, we always use the approximation: $\nabla L(C(\phi)) \approx M_\phi \cdot \nabla L(M_\phi \cdot \phi)$ for parameter ϕ, where M_ϕ is the Top-K mask of ϕ. We determine the value of the interpolation step ρ in C-SAM through grid search, using the same procedure as for CrAM, and we run each experiment for 3 different seeds. For fairness, we compare C-SAM with CrAM, and not CrAM$^+$, and for both we use sparsified gradients.

The results in Table 5.14 show that C-SAM can be more robust at higher sparsity levels, but with the cost of an accuracy drop for the dense models; for example, for C-SAM-k70, the test accuracy of the dense model, as well as at lower sparsity levels ($\leq 70\%$) is 1% below that obtained with CrAM-k70. One important aspect is that the dense model can be improved using CrAM$^+$ with no additional cost, whereas for C-SAM such a modification would incur a computational overhead, as each optimization step would require three instead of two forward-backward passes through the model.

Method	Dense	Sparsity				
		50%	60%	70%	80%	90%
CrAM-k50	92.8 ± 0.1	92.8 ± 0.1	92.7 ± 0.0	92.0 ± 0.1	89.7 ± 0.2	73.5 ± 2.4
C-SAM-k50	92.6 ± 0.3	92.6 ± 0.3	92.5 ± 0.3	92.0 ± 0.2	90.6 ± 0.2	81.0 ± 1.0
CrAM-k70	92.4 ± 0.2	92.4 ± 0.2	92.4 ± 0.2	92.3 ± 0.2	91.6 ± 0.1	81.1 ± 1.3
C-SAM-k70	91.4 ± 0.0	91.4 ± 0.0	91.4 ± 0.0	91.3 ± 0.1	91.0 ± 0.1	85.3 ± 0.5
CrAM-Multi	92.6 ± 0.2	92.5 ± 0.2	92.4 ± 0.4	92.3 ± 0.2	91.9 ± 0.2	90.5 ± 0.2
C-SAM-Multi	92.5 ± 0.2	92.5 ± 0.1	92.6 ± 0.2	92.5 ± 0.2	92.1 ± 0.2	91.0 ± 0.2

Table 5.14: (CIFAR-10/ResNet20) Comparison between CrAM and C-SAM, showing the test accuracy for the dense models and sparse models after one-shot pruning. We report the best value between the accuracy before and after BNT with 1000 training samples.

Importance of extra-gradient step. Furthermore, we explore the importance of the extra-gradient step in the CrAM update. Notably, we investigate whether not using the extra-gradient achieves a similar effect to CrAM training. The removal of the extra gradient step would correspond to the following equation:

$$\text{Top-K:} \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(C(\boldsymbol{\theta}_t)) . \tag{5.8}$$

Since the compression we use in our experiments is the Top-K sparsification, we refer to this update as "Top-K". This has been previously studied in [LSB+20], where the dense gradients, computed with respect to the sparse parameters, are used in the model updated. Generally, we have experienced training instability using this version of the update, particularly at high sparsity. However, we are able to substantially improve both the training stability and overall quality of the resulting models, by also incorporating the optimization of the dense model (similar to CrAM+), and by using sparsified gradients for the sparsified parameters. These changes resulted in an update close to CrAM+ and of the same computational complexity. This new update, which will be referred to as Top-K+ is the following:

$$\text{Top-K}^+: \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta(\nabla L(\boldsymbol{\theta}_t) + M_{\widetilde{\boldsymbol{\theta}}_t} \cdot \nabla L(\widetilde{\boldsymbol{\theta}}_t)), \tag{5.9}$$

where $\widetilde{\boldsymbol{\theta}}_t = C(\boldsymbol{\theta}_t)$ and $M_{\widetilde{\boldsymbol{\theta}}_t}$ is its mask after applying Top-K.

The comparison between CrAM+ and Top-K+ is illustrated in Table 5.15. The results show that dense CrAM+ models tend to have higher accuracy, and in general CrAM+ is more robust to one-shot pruning at high sparsity. For example, CrAM+ models trained with 50% or 70% sparsity and pruned one-shot to 80% and 90% sparsity achieve higher accuracy after BNT, compared to the corresponding sparse Top-K+ models.

Method	Dense	Sparsity				
		50%	60%	70%	80%	90%
CrAM+-k50	93.1 ± 0.1	93.1 ± 0.1	93.0 ± 0.1	92.3 ± 0.2	89.2 ± 0.2	71.1 ± 1.5
Top-K+-k50	92.7 ± 0.1	92.6 ± 0.0	92.6 ± 0.1	91.6 ± 0.1	86.5 ± 0.3	56.7 ± 1.0
CrAM+-k70	92.8 ± 0.3	92.7 ± 0.2	92.7 ± 0.1	92.7 ± 0.0	91.9 ± 0.	80.8 ± 1.4
Top-K+-k70	92.7 ± 0.1	92.4 ± 0.2	92.3 ± 0.2	92.4 ± 0.1	91.0 ± 0.3	72.6 ± 2.8
CrAM+-Multi	93.2 ± 0.1	93.2 ± 0.1	93.0 ± 0.1	92.8 ± 0.2	92.4 ± 0.1	90.1 ± 0.2
Top-K+-Multi	92.5 ± 0.1	92.4 ± 0.1	92.3 ± 0.1	92.2 ± 0.2	91.7 ± 0.2	90.0 ± 0.2

Table 5.15: (CIFAR-10/ResNet20) Comparison between CrAM+ and Top-K+. Test accuracy for the dense models and sparse models after one-shot pruning. For all sparse results we report the best value between the accuracy before and after BNT with 1000 training samples.

Conclusion. The comparison between CrAM and alternative updates, such as C-SAM or Top-K, shows that all methods can achieve good results with one-shot pruning. In particular, one-shot pruned C-SAM models tend to have a higher accuracy, compared to the corresponding CrAM models, in particular at high sparsity. However, the CrAM update allows for more flexibility, for example by incorporating for free the explicit optimization of the dense model. Overall, we conclude that CrAM-trained models achieve the best trade-off between preserving (or improving) the dense model accuracy, while having good performance after one-shot pruning, at different sparsity levels.

5.7 Conclusions and Future Work

In this chapter, we proposed a new method for training neural networks, CrAM, which results in models that are both highly accurate, and easily-compressible. Our extensive experimental analysis on large scale image classification (ImageNet/ResNets) and language modelling (SQuADv1.1/BERT-base) focuses on compression methods based on pruning, and shows that CrAM models can be pruned one-shot at a wide range of sparsity levels, while resulting in

sparse models that are competitive with existing gradual pruning methods. Furthermore, we show that one-shot pruned CrAM models can transfer better to downstream tasks, compared to some of the existing pruning methods. While we focus on pruning as the main compression operator, we also give encouraging evidence that the CrAM update can be successfully adapted to other compression projections, such as quantization, and we plan to investigate this more closely in future work. Furthermore, we would like to explore whether prolonged CrAM-training would further enhance both the performance of the resulting dense model, as well as its robustness to one-shot compression. We also note that CrAM could potentially be used in conjunction with other pruning methods, such as AC/DC described in Chapter 3, to boost the predictive performance of the resulting sparse models. Finally, we are interested in leveraging in the CrAM update different methods developed for reducing the computational complexity of SAM, in order to improve the efficiency of our method.

CHAPTER 6

Discussion and Future Work

In this thesis we studied pruning methods for deep learning models, with a focus on obtaining sparse models that preserve accuracy, compared to the dense baseline, while also exploring aspects related to their generalization properties and efficiency.

Thesis Summary

In Chapter 3 we proposed a method for co-training sparse and dense models, that can result in both highly-accurate sparse models, and dense models that do not lose accuracy with respect to the baseline, all at a lower theoretical computational cost compared to regular dense training. The particular way our models were trained allowed us to more easily explore the differences between sparse and dense models, at prediction level, and also to better understand how prone they each are to memorization. Thus, we showed that our sparse-dense model pairs agreed on more samples, including incorrect predictions, compared to gradual magnitude pruning and its corresponding parent dense model. Moreover, our analysis suggests that sparse models are less prone to memorizing randomly labelled data. Additionally, we provided theoretical guarantees in terms of convergence to a sparse local minimum, for the algorithm we used in practice.

While sparse and dense models can have very similar performance in terms of test accuracy, and they can also agree on their predictions on a large fraction of the samples, it is not clear how well they each generalize to changes in the data distribution. We attempted to answer this question in Chapter 4, where we investigated the behaviour of sparse models obtained from different pruning methods, in a transfer learning scenario, compared to the dense baseline. Our findings suggest that sparse models, in particular those obtained from sparse regularization methods, produce features that can generalize better than the dense ones, in a linear finetuning setup. In comparison, when performing full finetuning on downstream tasks, the transfer performance tends to correlate with the accuracy on the original task, and progressive sparsification methods that prune from pre-trained models tend to generalize better than regularization-based pruning methods. We also investigated potential factors that could lead to the different behavior of sparse models in regards to their performance on transfer. Our study offers practical guidelines on which sparse models are more appropriate to be used for finetuning on a downstream task, depending on the type of transfer, as well as on the difficulty of the task.

Inspired by the observations regarding the transferability of sparse neural networks to different data distributions, we further investigated how to better facilitate the deployment of sparse

models to smaller devices. With this motivation in mind, we proposed in Chapter 5 a method, which we call CrAM, for training accurate dense neural networks that are prunable in one step, after training, to different sparsity levels. On one hand, this would reduce the computational challenge of repeating the training cycle for each individual target sparsity, which is the current standard [FC19, RFC20, EGM+20, KRS+20, PIVA21], and would enable more flexibility when using sparse models on edge devices, as it would only require storing a single model. The method we proposed is inspired by others [FKMN21] designed to reduce the overall loss "sharpness" and improve generalization. Indeed, our method can result in a small generalization advantage compared to regular training. Moreover, the sparse models we obtain one-shot can be competitive with those obtained from current state-of-the-art pruning methods [LSB+20, KRS+20], and can also transfer well to different downstream tasks, in a full finetuning scenario.

Thesis Contributions

Therefore, this thesis focused on two important themes: more efficient ways of obtaining sparse neural networks, without sacrificing predictive performance, and studying the generalization properties of sparse models, in particular with respect to changes in the data distribution. We believe that the contributions presented in this thesis have helped improve the state of the art regarding training sparse models, and have also enabled a better understanding regarding the differences between pruning methods, beyond model accuracy.

Specifically, the AC/DC method presented in Chapter 3 achieves state-of-the-art results for models at different sparsity levels on ImageNet/ResNet50, when performing extended training. This has important practical implications, as the highly accurate sparse models obtained with AC/DC could be successfully used for sparse transfer on edge devices. Furthermore, to the best of our knowledge, we were the first to show that models can be successfully trained to be compressible, while also preserving or improving their accuracy with respect to the dense baseline. Namely, the CrAM method we proposed in Chapter 5 achieves state-of-the-art results for one-shot pruning post-training, without additional finetuning. Lastly, we provided the first in-depth study regarding the differences between pruning methods in a transfer learning scenario, and we showed that despite the similarities in upstream model accuracy, the choice of the pruning method can incur different effects in terms of transfer performance.

Future Directions

While we have proposed and analyzed pruning methods with the efficiency and generalization goals in mind, we acknowledge that there are other areas along these two directions that we have mentioned, but not explicitly explored in this thesis. In what follows, we discuss in more detail potential directions for future work.

Leveraging Sparsity for Accelerated Training. In Chapter 3 we presented a method that can train sparse models at a fraction of the cost for the dense baseline. Moreover, other works [EGM+20, JPR+20] have proposed methods that can perform training only in the sparse support, and therefore should lead to computational savings. Also, we discussed in Chapter 4 the benefits of performing sparse finetuning, from a computational point of view. However, the acceleration in total training time from sparsity is thus far only theoretical, and researchers rely on approximate metrics, such as FLOPs, to estimate the total compute time. We therefore believe it is important to concentrate more effort in the direction of developing pruning methods that can achieve practical speed-up during training. Possible solutions would

be a version of the backpropagation algorithm that can be more easily adapted to sparse matrices multiplication [NPI+23] or developing methods for structured sparsity that can achieve a better trade-off between accuracy and speed-up. Structured sparsity, in particular, has a good potential for speed-up, since it effectively decreases the size of the network. However, as we have already discussed in Section 2.2.3, current methods for structured sparsity do not yet achieve a good trade-off between accuracy and speed-up, and there are many difficulties that arise due to, for example, solving the correlations between residuals, as well as consecutive layers, when entire channels are removed. Therefore, we believe structured sparsity is a direction with good practical potential, and plenty of room for improvements.

Designing Pruning Methods Amenable to Distribution Shifts. In Chapter 4 we showed that sparse models obtained from current state-of-the-art pruning methods can achieve good accuracy when performing sparse finetuning on downstream tasks. Despite these encouraging findings, however, we note that there is a substantial performance gap with respect to the dense baseline at higher sparsity levels (e.g. $> 90\%$). One potential solution to narrowing this gap would be to adapt the pruning mask to the downstream task; however, this would come with additional costs, as it would imply more hyperparameter tuning, tailored for each individual task. A different, more streamlined approach would be to design pruning methods that can be guaranteed to generalize well across changes in the input data distribution. One source of inspiration for achieving such a goal would be the meta-learning approach [FAL17], where models are trained on many different small tasks, with the purpose of learning to quickly generalizing to new tasks; a particular case where meta-learning methods were proven successful is few-shot classification, where the model is expected to generalize to new classes after having seen only a small number of reference samples. More recently, the Meta-Dataset [TZD+20] was proposed, consisting of multiple heterogeneous datasets, with realistic class imbalances; such a dataset, together with meta-learning methods developed for few-shot learning could represent a starting point for our goal.

Data-free Pruning. In addition to these directions, we believe that there are other orthogonal ones of particular interest. For example, data-free pruning, or post-training pruning, which essentially refers to methods for obtaining sparse models without access to the training set, is of great practical importance, particularly in situations where the models are extremely large, or they were trained on proprietary data. Several works have shown that quantization can be very successful in this regime [FA22a, FAHA23], and there have already been attempts for pruning [HCI+21, FA22a, FA23]. Moreover, the CrAM method we proposed in Chapter 5 can be viewed as related to these methods, as it gives a model that can be pruned without additional finetuning. Despite these advancements, however, there is still a considerable gap compared to traditional data-aware pruning strategies, in particular for high sparsity levels.

The Effects of Pruning Beyond Accuracy. Another interesting direction relates to understanding the effects of pruning beyond test accuracy. Several works have studied how pruning affects under-represented groups in the training set [LBC+21, HCC+19, HMC+20], but they have thus far focused on progressive sparsification methods, and a more thorough exploration is required to understand how these effects hold for other types of pruning methods. Subsequently, this understanding could be used towards designing novel pruning methods with fairness objectives in mind. In addition to bias and fairness, we believe it is important to study the role pruning has in better understanding the generalization of neural networks, and how it relates to newly observed phenomena, such as the double-descent curve [BHMM19].

While recent work has hinted that a similar double descent curve could occur for pruned models [CLOT21], the analysis is performed on random feature regression tasks, and it is not clear how it would generalize in a more practical setup.

In conclusion, in this thesis we have investigated important aspects related to efficiency and generalization of sparse neural networks, and have proposed new methods for obtaining accurate sparse models. We hope that the results presented, together with the methods developed, will have a positive practical impact towards the deployment of sparse models to edge devices, and are complementary to other works investigating generalization properties of sparse neural networks.

Bibliography

[AF22] Maksym Andriushchenko and Nicolas Flammarion. Towards understanding sharpness-aware minimization. In *International Conference on Machine Learning (ICML)*, 2022.

[AGNZ18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning (ICML)*, 2018.

[AHJ+18] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. The convergence of sparsified gradient methods. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[AZLS19] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning (ICML)*, 2019.

[BBLP13] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*, 2015.

[BD08] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[BFT17] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[BGMMS21] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

[BGVG14] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision (ECCV)*, 2014.

[BHK+20] 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.

[BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[BKML18] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *International Conference on Learning Representations (ICLR)*, 2018.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[BLG+19] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *International Conference on Learning Representations (ICLR)*, 2019.

[BLG+22] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sensitivity-informed provable pruning of neural networks. *SIAM Journal on Mathematics of Data Science*, 4(1):26–45, 2022.

[BLL+14] Thomas Berg, Jiongxin Liu, Seung Woo Lee, Michelle L. Alexander, David W. Jacobs, and Peter N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[BMBE20] Brian Bartoldson, Ari Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[BZR+22] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[BZXL19] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. YOLACT: Real-time Instance Segmentation. In *International Conference on Computer Vision (ICCV)*, 2019.

[Cat07] Olivier Catoni. PAC-Bayesian Supervised Classification. *Lecture Notes-Monograph Series. IMS*, 1277, 2007.

[CFC+20] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained BERT networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[CFC+21] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[CGW+19] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *International Conference on Learning Representations (ICLR)*, 2019.

[CHS+16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[CJD+21] Tianyi Chen, Bo Ji, Tianyu Ding, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only train once: A one-shot neural network training and pruning framework. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[CLOT21] Xiangyu Chang, Yingcong Li, Samet Oymak, and Christos Thrampoulidis. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[CMK+14] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[CT06] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[Dan12] John M Danskin. *The theory of max-min and its application to weapons allocation problems*, volume 5. Springer Science & Business Media, 2012.

[DBC+21] James Diffenderfer, Brian Bartoldson, Shreya Chaganti, Jize Zhang, and Bhavya Kailkhura. A winning hand: Compressing deep networks can improve out-of-distribution robustness. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations (ICLR)*, 2021.

[DCLT19] J. 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.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[Dee21] DeepSparse. NeuralMagic DeepSparse Inference Engine, 2021.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12(7), 2011.

[DJV+14] 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 (ICML)*, 2014.

[DPBB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning (ICML)*, 2017.

[DR17] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *Uncertainty in Artificial Intelligence (UAI)*, 2017.

[DSD+13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2013.

[DYF+22] Jiawei Du, Hanshu Yan, Jiashi Feng, Joey Tianyi Zhou, Liangli Zhen, Rick Siow Mong Goh, and Vincent YF Tan. Efficient sharpness-aware minimization for improved training of neural networks. *International Conference on Learning Representations (ICLR)*, 2022.

[DYT+21] Josip Djolonga, Jessica Yung, Michael Tschannen, Rob Romijnders, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Matthias Minderer, Alexander D'Amour, Dan Moldovan, et al. On robustness and transferability of convolutional neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[DYY+19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[DZ19] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[DZF+22] Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent YF Tan, and Joey Tianyi Zhou. Sharpness-aware training for free. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[EDGS20] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[EGM+20] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, 2020.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

[FA22a] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[FA22b] Elias Frantar and Dan Alistarh. SPDY: Accurate pruning with speedup guarantees. *International Conference on Machine Learning (ICML)*, 2022.

[FA23] Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. *International Conference on Machine Learning (ICML)*, 2023.

[FAHA23] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate quantization for generative pre-trained transformers. *International Conference on Learning Representations (ICLR)*, 2023.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[FC19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[FDRC19] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[FDRC20] 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.

[FKA21] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-FAC: Efficient matrix-free approximations of second-order information. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[FKMN21] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *International Conference on Learning Representations (ICLR)*, 2021.

[FMI83] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

[Fou11] Simon Foucart. Hard thresholding pursuit: an algorithm for compressive sensing. *SIAM Journal on Numerical Analysis*, 49(6):2543–2563, 2011.

[Fou12] Simon Foucart. Sparse recovery algorithms: sufficient conditions in terms of restricted isometry constants. In *Approximation Theory XIII: San Antonio 2010*, pages 65–77. Springer, 2012.

[Fun89] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[GEH19] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[GHP06] Gregory Griffin, Alexander D. Holub, and Pietro Perona. The Caltech 256. *Caltech Technical Report*, 2006.

[GKD+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[GLQ+19] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. SGD: General Analysis and Improved Rates. *International Conference on Machine Learning (ICML)*, 2019.

[GMG+21] Sharath Girish, Shishira R Maiya, Kamal Gupta, Hao Chen, Larry S Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[Gra21] Graphcore. Graphcore Poplar SDK 2.0, 2021.

[HABN+21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research (JMLR)*, 2021.

[Hag94] Masafumi Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207 – 218, 1994. Backpropagation, Part IV.

[HCC+19] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248*, 2019.

[HCI+21] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find N: M transposable masks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[HLL+18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 2018.

[HMC+20] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058*, 2020.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[Hoy04] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research (JMLR)*, 5(9), 2004.

[HPN+17] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. DSD: Dense-sparse-dense training for deep neural networks. *International Conference on Learning Representations (ICLR)*, 2017.

[HPTD15] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[HS97a] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

[HS97b] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[HSW93] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

[HTFF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HZC+17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[IPA23] Eugenia Iofinova, Alexandra Peste, and Dan Alistarh. Bias in Pruned Vision Models: In-Depth Analysis and Countermeasures. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[IPKA22] Eugenia Iofinova, Alexandra Peste, Mark Kurtz, and Dan Alistarh. How well do sparse ImageNet models transfer? In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[JCR+22] Tian Jin, Michael Carbin, Daniel M Roy, Jonathan Frankle, and Gintare Karolina Dziugaite. Pruning's effect on generalization through the lens of training and regularization. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[JKRL09] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision (ICCV)*, 2009.

[JPR+20] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-K always sparse training. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[JTK14] Prateek Jain, Ambuj Tewari, and Purushottam Kar. On iterative hard thresholding methods for high-dimensional M-estimation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

[JYFY16] Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

[KA22] Eldar Kurtic and Dan Alistarh. GMP*: Well-Tuned Global Magnitude Pruning Can Outperform Most BERT-Pruning Methods. *arXiv preprint arXiv:2210.06384*, 2022.

[KB15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[KBZ+20] 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.

[KCN+22] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models. *arXiv preprint arXiv:2203.07259*, 2022.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KKG+20] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Bill Nell, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

[KKPC21] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. ASAM: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

[KMN+17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations (ICLR)*, 2017.

[KNS16] Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2016.

[KRJ+22] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *International Conference on Learning Representations (ICLR)*, 2022.

[KRS+20] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning (ICML)*, 2020.

[KSDFF13] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[KSL19] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better ImageNet models transfer better? In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[KSM+21] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning (ICML)*, 2021.

[KSW15] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[LAJ19] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. Structured pruning of neural networks with budget-aware regularization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[Lan86] Robert Lang. A note on the measurability of convex sets. *Archiv der Mathematik*, 47(1):90–92, 1986.

[LAT19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. SNIP: Single-shot network pruning based on connection sensitivity. *International Conference on Learning Representations (ICLR)*, 2019.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBC+21] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in Pruning: The Effects of Pruning Neural Networks beyond Test Accuracy. *Conference on Machine Learning and Systems (MLSys)*, 2021.

[LBL+20] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[LBOM12] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Conference on Neural Information Processing Systems (NeurIPS)*, 1990.

[LFP04] Fei-Fei Li, R. Fergus, and Pietro Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[LMC+22] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards efficient and scalable sharpness-aware minimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[LN89] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[LPW+17] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[LSB+20] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *International Conference on Learning Representations (ICLR)*, 2020.

[LSZ+19] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *International Conference on Learning Representations (ICLR)*, 2019.

[LUW17] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[LVdMJ+21] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[LWF+15] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[LWK18] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through ℓ_0 regularization. *International Conference on Learning Representations (ICLR)*, 2018.

[LXT+18] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[LZB20] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*, 2020.

[MAV17] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

[McA03] David McAllester. Simplified PAC-Bayesian margin bounds. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop (COLT/Kernel)*, 2003.

[MDL18] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *European Conference on Computer Vision (ECCV)*, 2018.

[Meh19] Rahul Mehta. Sparse transfer learning via winning lottery tickets. *arXiv preprint arXiv:1905.07785*, 2019.

[MJS22] Amirkeivan Mohtashami, Martin Jaggi, and Sebastian Stich. Masked training of neural networks with partial gradients. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[MLC+22] Lu Miao, Xiaolong Luo, Tianlong Chen, Wuyang Chen, Dong Liu, and Zhangyang Wang. Learning pruning-friendly networks via Frank-Wolfe: One-shot, any-sparsity, and no retraining. In *International Conference on Learning Representations (ICLR)*, 2022.

[MLP+21] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[MLY+21] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, et al. Non-structured dnn weight pruning—is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4930–4944, 2021.

[MMS+18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):1–12, 2018.

[MNA+18] 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. *International Conference on Learning Representations (ICLR)*, 2018.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MRK+13] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[MTK+17] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations (ICLR)*, 2017.

[MUK+22] Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. Factors of influence for transfer learning across diverse appearance domains and task types. *IEEE Transactions on Pattern Analysis & Machine Intelligence (TPAMI)*, 44(12):9298–9314, 2022.

[MXBS16] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[Nes83] Y. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on Machine Learning (ICML)*, 2010.

[NKB+21] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[NMAV17] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[NPI+23] Mahdi Nikdan, Tommaso Pegolotti, Eugenia Iofinova, Eldar Kurtic, and Dan Alistarh. SparseProp: Efficient sparse backpropagation for faster training of neural networks. *International Conference on Machine Learning (ICML)*, 2023.

[Nvi20] Nvidia Nvidia. A100 Tensor Core GPU architecture, 2020.

[NVI21] NVIDIA. NVIDIA Implementation of the Transformer-XL model architecture, 2021.

[NZ06] Maria-Elena Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[Ope22] Open AI. ChatGPT. `https://openai.com/blog/chatgpt/`, 2022. [Online; accessed 9 January 2023].

[PAL21] Alexandra Peste, Dan Alistarh, and Christoph H Lampert. SSSE: Efficiently Erasing Samples from Trained Machine Learning Models. *arXiv preprint arXiv:2107.03860*, 2021.

[PGD21] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[PGH+22] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.

[PIVA21] Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating Compressed/DeCompressed Training of Deep Neural Networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2013.

[Pol64] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[PVK+23] Alexandra Peste, Adrian Vladu, Eldar Kurtic, Christoph H. Lampert, and Dan Alistarh. CrAM: A Compression-Aware Minimizer. *International Conference on Learning Representations (ICLR)*, 2023.

[PVZJ12] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[PY21] Jeff Pool and Chong Yu. Channel permutations for N:M sparsity. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[RDN+22] 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.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[RFC20] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations (ICLR)*, 2020.

[RHW86a] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[RHW86b] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[RRSS19] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning (ICML)*, 2019.

[RSPS16] Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic Frank-Wolfe methods for nonconvex optimization. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[RWK+20] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[SA20] Sidak Pal Singh and Dan Alistarh. WoodFisher: Efficient second-order approximation for neural network compression. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[SCCS19] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding Top-K sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

[SGM20] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

[SIE+20] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust ImageNet models transfer better? *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[SM20] Alexander Shevchenko and Marco Mondelli. Landscape connectivity and dropout stability of SGD solutions for over-parameterized neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, 2013.

[SRASC14] 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.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Sut86] Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986.

[SWR20] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TAW+21] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, 2021.

[Teo07] Kwong Meng Teo. *Nonconvex robust optimization*. PhD thesis, Massachusetts Institute of Technology, 2007.

[TGL+22] Hugo Tessier, Vincent Gripon, Mathieu Léonardon, Matthieu Arzel, David Bertrand, and Thomas Hannagan. Leveraging structured pruning of convolutional neural networks. In *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, 2022.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[TZD+20] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *International Conference on Learning Representations (ICLR)*, 2020.

[Ult21] Ultralytics. Implementation of the YOLO V5 model architecture. *Available at* `https://github.com/ultralytics/yolov5`., 2021.

[Vap99] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[VDV23] Antoine Vanderschueren and Christophe De Vleeschouwer. Are straight-through gradients and soft-thresholding all you need for sparse training? In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. 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. Association for Computational Linguistics, 2020.

[WGFZ19] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning (ICML)*, 2019.

[WSM+18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, 2018.

[WWW+16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[WZL+21] Zhaofeng Wu, Ding Zhao, Qiao Liang, Jiahui Yu, Anmol Gulati, and Ruoming Pang. Dynamic sparsity neural networks for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2021.

[XHE+10] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. SUN Database: Large-scale Scene Recognition from Abbey to Zoo. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[YH19a] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

[YH19b] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[YJL+20] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. BigNAS: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision (ECCV)*, 2020.

[YLR+19] 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. In *International Conference on Learning Representations (ICLR)*, 2019.

[YLZ14] Xiaotong Yuan, Ping Li, and Tong Zhang. Gradient hard thresholding pursuit for sparsity-constrained optimization. In *International Conference on Machine Learning (ICML)*, 2014.

[YLZ+19] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *International Conference on Learning Representations (ICLR)*, 2019.

[YWL20] Huanrui Yang, Wei Wen, and Hai Li. DeepHoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *International Conference on Learning Representations (ICLR)*, 2020.

[YYX+19] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

[ZBH+17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR)*, 2017.

[ZBSC18] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

[ZLL+22] Yuxin Zhang, Mingbao Lin, Zhihang Lin, Yiting Luo, Ke Li, Fei Chao, Yongjian Wu, and Rongrong Ji. Learning Best Combination for Efficient N: M Sparsity. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[ZLLY19] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[ZMZ+21] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning N: M fine-grained structured sparse neural networks from scratch. *International Conference on Learning Representations (ICLR)*, 2021.

[ZSP22] Max Zimmer, Christoph Spiegel, and Sebastian Pokutta. Compression-aware training of neural networks using Frank-Wolfe. *arXiv preprint arXiv:2205.11921*, 2022.

[ZVA+19] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: a PAC-Bayesian compression approach. *International Conference on Learning Representations (ICLR)*, 2019.

[ZZL+22] Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. PLATON: Pruning Large Transformer Models with Upper Confidence Bound of Weight Importance. In *International Conference on Machine Learning (ICML)*, 2022.

APPENDIX A

Appendix for Chapter 3

A.1 Convergence Proofs

In this section we provide the convergence analysis for our algorithms. We prove the convergence of stochastic IHT, as stated in Theorem 3.3.1, and show as a corollary that under reasonable assumptions our implementation of AC/DC converges to a sparse minimizer.

A.1.1 Notations and Overview

Notations

We use the assumptions defined in Section 3.3.1. Since our analysis is based on bounding the progress made in a single iteration, to simplify notation we will generally use $\boldsymbol{\theta}$ to denote the current iterate, and $\boldsymbol{\theta}'$ to denote the iterate obtained after the IHT update:

$$\boldsymbol{\theta}' = T_k(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}),$$

where $g_{\boldsymbol{\theta}}$ is an unbiased stochastic estimator of the function $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ we want to minimize, i.e. $\mathbb{E}[g_{\boldsymbol{\theta}}|\boldsymbol{\theta}] = \nabla f(\boldsymbol{\theta})$. Moreover, as previously mentioned in Section 3.3.1, g has finite variance $\sigma > 0$: $\mathbb{E}[\|g_{\boldsymbol{\theta}} - \nabla f(\boldsymbol{\theta})\|^2] \leq \sigma^2$.

Additionally, we let S, S' to denote the support of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, respectively, and we use $\boldsymbol{\theta}^*$ to denote the assumed k^*-sparse minimizer. For an arbitrary vector $\mathbf{x}$, we use $\text{supp}(\mathbf{x})$ to denote its support, i.e. the set of coordinates where $\mathbf{x}$ is nonzero. Moreover, for any set $I \subset [N]$ of coordinates, we use the notation $\mathbf{x}_I$ to denote that for the vector $\mathbf{x}$, we use the corresponding I coordinates, and assume the others are zero. Furthermore, we may also refer to the minimizing value of f as $f^* = f(\boldsymbol{\theta}^*)$.

Overview

Our goal is to prove Theorem 3.3.1, which is our main theoretical result. We provide a proof in Section A.1.2, preceded by a set of auxiliary results, for which we also provide proofs. The main idea behind the algorithm is based on the standard IHT algorithm [BD08], which consists of alternating full gradient steps with pruning/truncation steps. Pruning to the largest k coordinates in absolute value, corresponds to a non-convex projection onto the set of k-sparse vectors. To ensure that the sparse projection does not have a major impact on the progress

made by the gradient descent steps, we assume a lower target sparsity, compared to the corresponding value of the sparse optimum. This corresponds to increasing the size of the support k to order $\Omega(k^* \cdot (\beta/\alpha)^2)$, where $\beta/\alpha > 1$ represents the "restricted condition number" of f.

Furthermore, we provide a corollary that offers guarantees for our practical AC/DC algorithm. This theoretical result takes into account the differences between AC/DC and the stochastic IHT algorithm presented in Section 3.3.1, namely the fact that AC/DC performs a sequence of several dense SGD steps before applying a single pruning step. We show that even with this change we can provide theoretical convergence bounds, although these bounds can be weaker than the baseline IHT method under our assumptions. The proof of this result is given in Section A.1.3.

In addition to the guarantees for stochastic IHT for functions with concentrated PL condition, we show that our results can be extended to functions without the CPL condition; in this case, we can prove convergence to a sparse nearly-stationary point. Furthermore, as a bonus, we can also derive convergence guarantees for stochastic IHT for strongly-convex functions. Both of these results can be found explained in detail in [PIVA21].

A.1.2 Stochastic IHT for Non-Convex Functions with Concentrated PL Condition

In this section we prove our main result presented in Theorem 3.3.1. We analyze stochastic IHT for a class of functions that satisfy a special version of the Polyak-Łojasiewicz (PL) condition [KNS16] which is standard in non-convex optimization; moreover, certain versions of PL were essential in several works analyzing the convergence of training methods for deep neural networks [LZB20, AZLS19]. Usually this condition says that small gradient norm implies closeness to optimum in function value. Here we use the stronger (r, α)-CPL condition, defined through Equation 3.4 from Section 3.3.1, which considers the norm of the gradient contributed by its largest coordinates in absolute value.

We prove strong convergence bounds for functions that satisfy the CPL condition. Compared to the classical Polyak-Łojasiewicz condition, this adds the additional assumption that most of the mass of the gradient is concentrated on a small subset of coordinates. This phenomenon has been witnessed in several instances, and is implicitly used in [LSB$^+$20].

Before proceeding with the main proof we provide a few useful lemmas.

Auxiliary Results

The first lemma we provide tells us that the distance from the current point after applying the Top-K operator, to the optimal sparse solution can be controlled, as long as we are in a point with lower sparsity compared to the sparse optimum. This lemma, which has been previously stated in [JTK14], is particularly important for our analysis and will be used in most of our proofs. We provide a slightly more general statement below.

Lemma A.1.1. *Let $\boldsymbol{\theta}^* \in \mathbb{R}^N$ be a k^*-sparse vector, and let $\boldsymbol{\theta} \in \mathbb{R}^N$ a k-sparse vector, with $k \geq k^*$. Then for any $n > k$ the following inequality holds:*

$$\frac{\|T_k(\boldsymbol{\theta}) - \boldsymbol{\theta}\|^2}{n-k} \leq \frac{\|\boldsymbol{\theta}^* - \boldsymbol{\theta}\|^2}{n-k^*}$$

Proof. It is easy to observe that the function $h(\boldsymbol{\theta}) = \frac{\|T_k(\boldsymbol{\theta})-\boldsymbol{\theta}\|^2}{n-k}$ is non-increasing. Indeed, a larger k in the Top-K operator can only decrease the ratio. Thus,

$$\frac{\|T_k(\boldsymbol{\theta}) - \boldsymbol{\theta}\|^2}{n-k} \leq \frac{\|T_{k^*}(\boldsymbol{\theta}) - \boldsymbol{\theta}\|^2}{n-k^*}$$

Furthermore, since the Top-K operator gives the closest sparse vector to $\boldsymbol{\theta}$, we also have that

$$\frac{\|T_{k^*}(\boldsymbol{\theta}) - \boldsymbol{\theta}\|^2}{n-k^*} \leq \frac{\|\boldsymbol{\theta}^* - \boldsymbol{\theta}\|^2}{n-k^*}$$

Combining these two inequalities completes the proof. □

Next, we provide a result showing a simplified version of the (t, β)-smoothness property.

Lemma A.1.2. *If $f : \mathbb{R}^N \to \mathbb{R}$ is a (t, β)-smooth function, then for any t-sparse vector $\boldsymbol{\delta}$, the following holds:*

$$f(\boldsymbol{\theta} + \boldsymbol{\delta}) \leq f(\boldsymbol{\theta}) + \frac{\beta}{2}\left\|\left(\frac{1}{\beta}\nabla f(\boldsymbol{\theta}) + \boldsymbol{\delta}\right)_{\mathsf{supp}(\boldsymbol{\delta})}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f(\boldsymbol{\theta})_{\mathsf{supp}(\boldsymbol{\delta})}\right\|^2 .$$

Proof. Using the (t, β)-smoothness we bound:

$$f(\boldsymbol{\theta} + \boldsymbol{\delta}) \leq f(\boldsymbol{\theta}) + \langle \nabla f(\boldsymbol{\theta}), \boldsymbol{\delta} \rangle + \frac{\beta}{2}\|\boldsymbol{\delta}\|^2$$

The RHS can be further rewritten as :

$$\begin{aligned} &f(\boldsymbol{\theta}) + \frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})\|^2 + \langle \nabla f(\boldsymbol{\theta}), \boldsymbol{\delta} \rangle + \frac{\beta}{2}\|\boldsymbol{\delta}\|^2 - \frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})\|^2 \\ &\quad = f(\boldsymbol{\theta}) + \frac{1}{2}\left\|\frac{1}{\sqrt{\beta}}\nabla f(\boldsymbol{\theta}) + \sqrt{\beta}\boldsymbol{\delta}\right\|^2 - \frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})\|^2 \\ &\quad = f(\boldsymbol{\theta}) + \frac{\beta}{2}\left\|\frac{1}{\beta}\nabla f(\boldsymbol{\theta}) + \boldsymbol{\delta}\right\|^2 - \frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})\|^2 . \end{aligned}$$

Since $\boldsymbol{\delta}$ is t-sparse, we notice that the contributions of the two terms $\frac{\beta}{2}\left\|\frac{1}{\beta}\nabla f(\boldsymbol{\theta}) + \boldsymbol{\delta}\right\|^2$ and $\frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})\|^2$ match on the coordinates where $\boldsymbol{\delta}$ is zero. Hence, we can reduce the result from the final two terms to the support of $\boldsymbol{\delta}$, which yields the desired conclusion. □

We require another lemma which will be very useful in the analysis.

Lemma A.1.3. *Let $\boldsymbol{\theta}, \boldsymbol{\delta} \in \mathbb{R}^N$ such that $\mathsf{supp}(\boldsymbol{\theta}) = S$, and let S', S^* be some arbitrary subsets, with $|S'| = |S| > |S^*|$. Furthermore, we assume that*

$$T_k(\boldsymbol{\theta} + \boldsymbol{\delta}) = (\boldsymbol{\theta} + \boldsymbol{\delta})_{S'} .$$

Then we have that

$$\left\|(\boldsymbol{\theta} + \boldsymbol{\delta})_{S\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 \leq \left\|(\boldsymbol{\theta} + \boldsymbol{\delta})_{Z\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S^*}\|^2 ,$$

where Z is a set such that $|Z \setminus S'| \leq 2|S^*|$.

Proof. For the proof, we first remind that $\boldsymbol{\theta}$ is zero everywhere except S. Therefore, we can rewrite the LHS of the inequality we want to prove as follows:

$$\begin{aligned}\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 &= \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(S^*\cup S)\setminus S'}\right\|^2 - \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S^*\setminus(S\cup S')}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 \\ &= \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(S^*\cup S)\setminus S'}\right\|^2 - \left\|\boldsymbol{\delta}_{S^*\setminus(S\cup S')}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 \\ &= \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(S^*\cup S)\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S^*\cup S\cup S'}\|^2 \\ &= \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(S^*\cup S)\setminus S'}\right\|^2 - \left\|\boldsymbol{\delta}_{S'\setminus(S^*\cup S)}\right\|^2 - \|\boldsymbol{\delta}_{S^*\cup S}\|^2 \ .\end{aligned}$$

Since $\boldsymbol{\theta}$ is zero outside S, we have that $\left\|\boldsymbol{\delta}_{S'\setminus(S^*\cup S)}\right\|^2 = \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S'\setminus(S^*\cup S)}\right\|^2$

Moreover, because $S^{'}$ is the support of the Top-K operator applied to $\boldsymbol{\theta}+\boldsymbol{\delta}$, we know that $\boldsymbol{\theta}+\boldsymbol{\delta}$ achieves the highest norm on the support $S^{'}$, compared to other subsets with the same number of elements. In particular, there exists a set $R \subseteq (S^* \cup S) \setminus S^{'}$ with $|R| = \left|S^{'} \setminus (S^* \cup S)\right|$ such that

$$\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S'\setminus(S^*\cup S)}\right\|^2 \geq \|(\boldsymbol{\theta}+\boldsymbol{\delta})_R\|^2$$

The existence of set R is guaranteed by the fact that

$$|(S^* \cup S) \setminus S'| \geq |S \setminus S'| = |S' \setminus S| \geq |S' \setminus (S^* \cup S)| = |R|$$

where for the equality in the middle we used that $|S| = \left|S^{'}\right|$.

Hence we obtain that

$$\begin{aligned}\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 &\leq \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(S^*\cup S)\setminus S'}\right\|^2 - \|(\boldsymbol{\theta}+\boldsymbol{\delta})_R\|^2 - \|\boldsymbol{\delta}_{S^*\cup S}\|^2 \\ &\leq \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{((S^*\cup S)\setminus S')\setminus R}\right\|^2 - \|\boldsymbol{\delta}_{S^*\cup S}\|^2\end{aligned}$$

We further note that $\left((S^* \cup S) \setminus S^{'}\right) \setminus R = ((S^* \cup S) \setminus R) \setminus S^{'}$ and since $R \subseteq (S^* \cup S) \setminus S'$, we have

$$\begin{aligned}|((S^* \cup S) \setminus S') \setminus R| &= |(S^* \cup S) \setminus S'| - |R| = |(S^* \cup S) \setminus S'| - |S' \setminus (S^* \cup S)| \\ &\leq (|S^*| + |S \setminus S'|) - (|S' \setminus S| - |S^*|) \\ &= 2\,|S^*| \ .\end{aligned}$$

Therefore, the conclusion holds for $Z = (S^* \cup S) \setminus R$. □

Based on the previous lemma we can derive the following useful corollary.

Corollary A.1.1. *Let $\boldsymbol{\theta}, \boldsymbol{\theta}^\star, \boldsymbol{\delta} \in \mathbb{R}^N$ such that* $\text{supp}\,(\boldsymbol{\theta}) = S$*, and let S', S^* be arbitrary subsets, with $|S'| = |S| > |S^*|$. Furthermore suppose that*

$$T_k\,(\boldsymbol{\theta}+\boldsymbol{\delta}) = (\boldsymbol{\theta}+\boldsymbol{\delta})_{S'} \ .$$

Then the following holds:

$$\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 \leq \frac{2\,|S^*| + |\text{supp}\,(\boldsymbol{\theta}^*)|}{|S'| - |S^*|} \cdot \|(\boldsymbol{\theta}+\boldsymbol{\delta})_T - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\delta}_{S^*}\|^2 \ ,$$

where T satisfies $|T| \leq 2\,|S^*| + |\text{supp}\,(\boldsymbol{\theta}^*)| + |S'|$ and $\text{supp}\,(\boldsymbol{\theta}^*) \subseteq T$.

Proof. Using Lemma A.1.3 we can write

$$\begin{aligned}\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{S\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S\cup S'}\|^2 \leq \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{Z\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S^*}\|^2 \leq \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{(Z\cup\mathsf{supp}(\boldsymbol{\theta}^*))\setminus S'}\right\|^2 - \|\boldsymbol{\delta}_{S^*}\|^2 \\ = \left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'} - (\boldsymbol{\theta}+\boldsymbol{\delta})_{S'}\right\|^2 - \|\boldsymbol{\delta}_{S^*}\|^2 \ .\end{aligned}$$

Based on the previous lemma, Z satisfies $|Z\setminus S| \leq 2|S^*|$. Thus, $|Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'| \leq 2|S^*| + |\mathsf{supp}(\boldsymbol{\theta}^*)| + |S'|$. Applying Lemma A.1.1 to the first term of the RHS, we obtain that

$$\begin{aligned}\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'} - (\boldsymbol{\theta}+\boldsymbol{\delta})_{S'}\right\|^2 &\leq \frac{|Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'| - |S'|}{|Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'| - |\mathsf{supp}(\boldsymbol{\theta}^*)|}\cdot\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'} - \boldsymbol{\theta}^*\right\|^2 \\ &\leq \frac{2|S^*| + |\mathsf{supp}(\boldsymbol{\theta}^*)|}{|S'| - |\mathsf{supp}(\boldsymbol{\theta}^*)|}\cdot\left\|(\boldsymbol{\theta}+\boldsymbol{\delta})_{Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'} - \boldsymbol{\theta}^*\right\|^2 \ .\end{aligned}$$

Therefore, the result holds for $T = Z\cup\mathsf{supp}(\boldsymbol{\theta}^*)\cup S'$.

□

A crucial step in the proof of Theorem 3.3.1 requires upper bounding the ℓ_2 distance to the closest global optimizer by the difference in function value. In general, it is known that this is implied by the Polyak-Łojasiewicz condition, and so it automatically holds for the stronger concentrated Polyak-Łojasiewicz condition. The following lemma, which also appears in [KNS16], gives us this upper bound. We partially reproduce the proof from [KNS16] for completeness.

Lemma A.1.4. *(From Polyak-Łojasiewicz to quadratic growth) Let $\alpha > 0$ and $f : \mathbb{R}^N \to \mathbb{R}$ be a function satisfying the Polyak-Łojasiewicz inequality*

$$\|\nabla f(\boldsymbol{\theta})\|^2 \geq \frac{\alpha}{2}(f(\boldsymbol{\theta}) - f^*)$$

where f^* is the optimal value of f. Then there exists a global minimizer $\boldsymbol{\theta}^*$ of f such that for any $\boldsymbol{\theta}$ the following holds:

$$f(\boldsymbol{\theta}) - f^* \geq \frac{\alpha}{8}\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2 \ .$$

Proof. We define the function $h(\boldsymbol{\theta}) = \sqrt{f(\boldsymbol{\theta}) - f^*}$, which has the following gradient

$$\nabla h(\boldsymbol{\theta}) = \frac{1}{2\sqrt{f(\boldsymbol{\theta}) - f^*}}\nabla f(\boldsymbol{\theta}) \ .$$

Using the PL condition we have

$$\|\nabla h(\boldsymbol{\theta})\|^2 = \frac{1}{4(f(\boldsymbol{\theta}) - f^*)}\cdot\|\nabla f(\boldsymbol{\theta})\|^2 \geq \frac{1}{4(f(\boldsymbol{\theta}) - f^*)}\cdot\frac{\alpha}{2}\cdot(f(\boldsymbol{\theta}) - f^*) = \frac{\alpha}{8} \ .$$

For an initial point $\boldsymbol{\theta}_0$, we consider the differential equation $\dot{\boldsymbol{\theta}} = -\nabla h(\boldsymbol{\theta})$. We see that this always decreases function value until it reaches some $\boldsymbol{\theta}_T$ for which $\nabla h(\boldsymbol{\theta}_T) = 0$. Using the PL inequality, $\boldsymbol{\theta}_T$ is a minimizer for f, i.e. $f(\boldsymbol{\theta}_T) = f^*$. Now we can write

$$\begin{aligned}h(\boldsymbol{\theta}_T) &= h(\boldsymbol{\theta}_0) + \int_0^T \left\langle \nabla h(\boldsymbol{\theta}_t), \dot{\boldsymbol{\theta}}_t \right\rangle dt = h(\boldsymbol{\theta}_0) + \int_0^T \langle \nabla h(\boldsymbol{\theta}_t), -\nabla h(\boldsymbol{\theta}_t)\rangle\, dt \\ &= h(\boldsymbol{\theta}_0) - \int_0^T \|\nabla h(\boldsymbol{\theta}_t)\|^2\, dt \ .\end{aligned}$$

Therefore,

$$h\left(\boldsymbol{\theta}_0\right)-h\left(\boldsymbol{\theta}_T\right)=\int_0^T\left\|\nabla h\left(\boldsymbol{\theta}_t\right)\right\|^2 dt \geq \sqrt{\frac{\alpha}{8}}\cdot\int_0^T\left\|\nabla h\left(\boldsymbol{\theta}_t\right)\right\| dt=\sqrt{\frac{\alpha}{8}}\cdot\int_0^T\left\|\dot{\boldsymbol{\theta}}_t\right\| dt \ ,$$

where for the inequality we used our lower bound on the norm of $\nabla h\left(\boldsymbol{\theta}\right)$. Finally, we use the fact that the last integral is larger than the total movement of $\boldsymbol{\theta}$ as it moves from $\boldsymbol{\theta}_0$ to $\boldsymbol{\theta}_T$. Thus,

$$\int_0^T\left\|\dot{\boldsymbol{\theta}}_t\right\| dt \geq \left\|\boldsymbol{\theta}_0-\boldsymbol{\theta}_T\right\| \ ,$$

Therefore, we obtain that $h\left(\boldsymbol{\theta}_0\right)-h\left(\boldsymbol{\theta}_T\right)\geq\sqrt{\frac{\alpha}{8}}\left\|\boldsymbol{\theta}_0-\boldsymbol{\theta}_T\right\|$. Since $\boldsymbol{\theta}_T$ is a minimizer with $f(\boldsymbol{\theta}_T)=f^*$, $h(\boldsymbol{\theta}_0)-h(\boldsymbol{\theta}_T)=\sqrt{f(\boldsymbol{\theta})-f^*}$, which enables us to conclude that

$$f\left(\boldsymbol{\theta}_0\right)-f^*\geq\frac{\alpha}{8}\left\|\boldsymbol{\theta}_0-\theta_T\right\|^2 \ .$$

□

Finally, we provide one more auxiliary lemma.

Lemma A.1.5. *Let $\sigma>0$ and a function $f:\mathbb{R}^N\longrightarrow\mathbb{R}$ for which we have unbiased stochastic gradients $g_{\boldsymbol{\theta}}$ with bounded variance. Namely:*

$$\mathbb{E}[g_{\boldsymbol{\theta}}|\boldsymbol{\theta}]=\nabla f(\boldsymbol{\theta}),$$

and

$$\mathbb{E}[\|g_{\boldsymbol{\theta}}-\nabla f(\boldsymbol{\theta})\|^2]\leq\sigma^2 .$$

Then for any $\gamma\in\mathbb{R}^*$, any vector $\mathbf{a}\in\mathbb{R}^N$ and any subset $S\subseteq[N]$ of coordinates, the following inequalities hold:

$$\left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2\leq\mathbb{E}\left[\left\|\left(\gamma g_{\boldsymbol{\theta}}+\mathbf{a}\right)_S\right\|^2\middle|\boldsymbol{\theta}\right]\leq\left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2+\sigma^2\gamma^2 \ .$$

Proof. First, we expand the norm of $\left(g_{\boldsymbol{\theta}}+\mathbf{a}\right)_S$ under the expectation with respect to $\boldsymbol{\theta}$:

$$\begin{aligned}
\mathbb{E}\left[\left\|\left(\gamma g_{\boldsymbol{\theta}}+\mathbf{a}\right)_S\right\|^2\middle|\boldsymbol{\theta}\right] &= \mathbb{E}\left[\left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S+\left(\gamma g_{\boldsymbol{\theta}}-\gamma\nabla f\left(\boldsymbol{\theta}\right)\right)_S\right\|^2\middle|\boldsymbol{\theta}\right] \\
&= \left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2+\mathbb{E}\left[\left\|\left(\gamma g_{\boldsymbol{\theta}}-\gamma\nabla f\left(\boldsymbol{\theta}\right)\right)_S\right\|^2\middle|\boldsymbol{\theta}\right] \\
&+\mathbb{E}\left[2\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S^T\left(\gamma g_{\boldsymbol{\theta}}-\gamma\nabla f\left(\boldsymbol{\theta}\right)\right)_S\middle|\boldsymbol{\theta}\right] \\
&= \left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2+\mathbb{E}\left[\left\|\left(\gamma g_{\boldsymbol{\theta}}-\gamma\nabla f\left(\boldsymbol{\theta}\right)\right)_S\right\|^2\middle|\boldsymbol{\theta}\right] \\
&\leq \left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2+\sigma^2\gamma^2 \ .
\end{aligned}$$

Based on the final equality, we immediately get that $\mathbb{E}\left[\left\|\left(\gamma g_{\boldsymbol{\theta}}+\mathbf{a}\right)_S\right\|^2\middle|\boldsymbol{\theta}\right]\geq\left\|\left(\gamma\nabla f\left(\boldsymbol{\theta}\right)+\mathbf{a}\right)_S\right\|^2$, which completes the proof. □

After these preparations, we can now proceed with the main proof.

Proof of Theorem 3.3.1

In what follows, we remind that we use S, S' to denote the support of the current iterate $\boldsymbol{\theta}$ and the next $\boldsymbol{\theta}'$, respectively. Furthermore, we denote $k = |S|$ and $k^* = |\text{supp}(\boldsymbol{\theta}^*)|$, with $k > k^*$. Since we apply the Top-K operator with the same sparsity k, we note that $|S| = |S'|$.

Before proceeding with the proof, we re-state the theorem.

Theorem 3.3.1. *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function satisfying previous assumptions (1)-(4), with a k^*-sparse minimizer $\boldsymbol{\theta}^*$. Let $\beta > \alpha > 0$ be parameters, let $k = C \cdot k^* \cdot (\beta/\alpha)^2$ for some appropriately chosen constant C, and suppose that f is $(2k+3k^*, \beta)$-smooth and (k^*, α)-CPL.*

For initial parameters $\boldsymbol{\theta}_0$ and precision $\epsilon > 0$, given access to stochastic gradients with variance σ, stochastic IHT (3.3) converges in $O\left(\frac{\beta}{\alpha} \cdot \ln \frac{f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*)}{\epsilon}\right)$ iterations to a point $\boldsymbol{\theta}$ with $\|\boldsymbol{\theta}\|_0 \leq k$, such that

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}\right) - f\left(\boldsymbol{\theta}^*\right)\right] \leq \epsilon + \frac{16\sigma^2}{\alpha}.$$

Proof. First, we use the smoothness assumption of f and we apply Lemma A.1.2 for the perturbation $\boldsymbol{\delta} = T_k\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right) - \boldsymbol{\theta}$, and use the fact that $\text{supp}\left(\boldsymbol{\delta}\right) \subseteq S \cup S'$. This gives us:

$$\begin{aligned}
f\left(\boldsymbol{\theta}'\right) &\leq f\left(\boldsymbol{\theta}\right) + \frac{\beta}{2}\left\|\left(\frac{1}{\beta}\nabla f\left(\boldsymbol{\theta}\right) + \left(T_k\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right) - \boldsymbol{\theta}\right)\right)_{S\cup S'}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f\left(\boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 \\
&= f\left(\boldsymbol{\theta}\right) + \frac{\beta}{2}\left\|\left(T_k\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right) - \left(\boldsymbol{\theta} - \frac{1}{\beta}\nabla f\left(\boldsymbol{\theta}\right)\right)\right)_{S\cup S'}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f\left(\boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 \\
&\leq f\left(\boldsymbol{\theta}\right) + \beta\left(\left\|\left(T_k\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right) - \left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right)\right)_{S\cup S'}\right\|^2 + \left\|\left(\eta g_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f\left(\boldsymbol{\theta}\right)\right)_{S\cup S'}\right\|^2\right) - \frac{1}{2\beta}\left\|\nabla f\left(\boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 \\
&= f\left(\boldsymbol{\theta}\right) + \beta\left\|\left(\boldsymbol{\theta} - \eta g_{\theta}\right)_{S\setminus S'}\right\|^2 + \beta\left\|\left(\eta g_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f\left(\boldsymbol{\theta}\right)\right)_{S\cup S'}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f\left(\boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 \ .
\end{aligned}$$

For the inequality we used that $\|\mathbf{a}+\mathbf{b}\|^2 \leq 2(\|\mathbf{a}\|^2 + \|\mathbf{b}\|^2)$, while for the final equality we used that the support of $\boldsymbol{\theta}'$ is S', which is obtained after applying the Top-K operator to $\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}$.

Next, we apply Corollary A.1.1 to $\boldsymbol{\theta}$, $-\eta g_{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}^*$, and where we define $S^* = T_{k^*}(\nabla f(\theta))$, i.e. $|S^*| = |\text{supp}(\boldsymbol{\theta}^*)| = k^*$. The corollary tells us that there exists a subset of indices T with $|T| \leq 3k^* + k$, such that the following holds:

$$\begin{aligned}
\left\|\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right)_{S\setminus S'}\right\|^2 &\leq \frac{2\left|S^*\right| + \left|\text{supp}\left(\boldsymbol{\theta}^*\right)\right|}{\left|S'\right| - \left|\text{supp}\left(\boldsymbol{\theta}^*\right)\right|} \cdot \left\|\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right)_T - \boldsymbol{\theta}^*\right\|^2 - \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S^*}\right\|^2 + \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S\cup S'}\right\|^2 \\
&= \frac{3k^*}{k-k^*} \cdot \left\|\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right)_T - \boldsymbol{\theta}^*\right\|^2 - \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S^*}\right\|^2 + \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S\cup S'}\right\|^2 \ ,
\end{aligned}$$

Therefore, using the previous inequalities, we have:

$$\begin{aligned}
f\left(\boldsymbol{\theta}'\right) \leq f\left(\boldsymbol{\theta}\right) &+ \beta\left(\frac{3k^*}{k-k^*} \cdot \left\|\left(\boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}\right)_T - \boldsymbol{\theta}^*\right\|^2 - \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S^*}\right\|^2 + \left\|\eta\left(g_{\boldsymbol{\theta}}\right)_{S\cup S'}\right\|^2\right) \\
&+ \beta\left\|\left(\eta g_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f\left(\boldsymbol{\theta}\right)\right)_{S\cup S'}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f\left(\boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 \ .
\end{aligned}$$

We have the following:

$$\begin{aligned}\|(\boldsymbol{\theta}-\eta g_{\boldsymbol{\theta}})_T-\boldsymbol{\theta}^*\|^2 &\leq 2\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_T-\boldsymbol{\theta}^*\|^2+2\|(\eta\nabla f(\boldsymbol{\theta})-\eta g_{\boldsymbol{\theta}})_T\|^2\\ &\leq 2\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_T-\boldsymbol{\theta}^*\|^2+2\eta^2\|\nabla f(\boldsymbol{\theta})-g_{\boldsymbol{\theta}}\|^2\end{aligned}$$

Taking expectations conditioned on $\boldsymbol{\theta}$, we obtain that

$$\mathbb{E}\left[\|(\boldsymbol{\theta}-\eta g_{\boldsymbol{\theta}})_T-\boldsymbol{\theta}^*\|^2|\boldsymbol{\theta}\right]\leq 2\mathbb{E}\left[\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_T-\boldsymbol{\theta}^*\|^2\right]+2\eta^2\sigma^2$$

Furthermore,

$$\begin{aligned}&\beta\|\eta(g_{\boldsymbol{\theta}})_{S\cup S'}\|^2+\beta\left\|\left(\eta g_{\boldsymbol{\theta}}-\frac{1}{\beta}\nabla f(\boldsymbol{\theta})\right)_{S\cup S'}\right\|^2-\frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})_{S\cup S'}\|^2=\\ &=2\beta\eta^2\|(g_{\boldsymbol{\theta}})_{S\cup S'}\|^2+\frac{1}{2\beta}\|\nabla f(\boldsymbol{\theta})_{S\cup S'}\|^2-2\eta(g_{\boldsymbol{\theta}})_{S\cup S'}^T\cdot\nabla f(\boldsymbol{\theta})_{S\cup S'}\\ &=2\beta\left\|\left(\eta g_{\boldsymbol{\theta}}-\frac{1}{2\beta}\nabla f(\boldsymbol{\theta})\right)_{S\cup S'}\right\|^2\leq 2\beta\left\|\eta g_{\boldsymbol{\theta}}-\frac{1}{2\beta}\nabla f(\boldsymbol{\theta})\right\|^2\end{aligned}$$

Taking expectations conditioned on $\boldsymbol{\theta}$ and using Lemma A.1.5, we have

$$2\beta\mathbb{E}\left[\left\|\eta g_{\boldsymbol{\theta}}-\frac{1}{2\beta}\nabla f(\boldsymbol{\theta})\right\|^2\Big|\boldsymbol{\theta}\right]\leq 2\beta\left\|\eta\nabla f(\boldsymbol{\theta})-\frac{1}{2\beta}\nabla f(\boldsymbol{\theta})\right\|^2+2\beta\eta^2\sigma^2$$

Setting $\eta=\frac{1}{2\beta}$, we further obtain $2\beta\mathbb{E}\left[\left\|\eta g_{\boldsymbol{\theta}}-\frac{1}{2\beta}\nabla f(\boldsymbol{\theta})\right\|^2\Big|\boldsymbol{\theta}\right]\leq 2\beta\eta^2\sigma^2$

Putting everything together, we obtain that:

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}')-f(\boldsymbol{\theta}^*)|\boldsymbol{\theta}]\leq f(\boldsymbol{\theta})-f(\boldsymbol{\theta}^*)+\frac{6\beta k^*}{k-k^*}\mathbb{E}\left[\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_T-\boldsymbol{\theta}^*\|^2\right]+\\ +\frac{6\beta k^*}{k-k^*}\eta^2\sigma^2-\beta\eta^2\|\nabla f(\boldsymbol{\theta})_{S^*}\|^2+2\beta\eta^2\sigma^2\end{aligned}$$

From Corollary A.1.1 we know that $\text{supp}(\boldsymbol{\theta}^*)\subseteq T$. Therefore, we can obtain an upper-bound by increasing the support of $\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta})$.

$$\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_T-\boldsymbol{\theta}^*\|^2\leq\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_{T\cup S}-\boldsymbol{\theta}^*\|^2$$

Since, f is (k^*,α)-CPL, it is also automatically PL. Then, we can apply Lemma A.1.4 to obtain the upper bound

$$\|(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_{T\cup S}-\boldsymbol{\theta}^*\|^2\leq\frac{8}{\alpha}\left(f\left((\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta}))_{T\cup S}\right)-f(\boldsymbol{\theta}^*)\right)$$

Next, we know from the assumptions that f is $(3k^*+2k)$-smooth. Since $|T|\leq 3|S^*|+|S'|$ (from Corollary A.1.1), we have that $|T\cup S|\leq 3k^*+2k$. Therefore, we can directly apply the smoothness assumption, which, together with the fact that $\eta=\frac{1}{2\beta}$, gives us that:

$$f(\boldsymbol{\theta}-\eta\nabla f(\boldsymbol{\theta})_{T\cup S})-f(\boldsymbol{\theta}^*)\leq f(\boldsymbol{\theta})-f(\boldsymbol{\theta}^*)\qquad(*)$$

Moreover, we can directly apply the (k^*,α)-CPL condition to bound $\|\nabla f(\boldsymbol{\theta})_{S^*}\|^2\geq\frac{\alpha}{2}\left(f(\boldsymbol{\theta})-f(\boldsymbol{\theta}^*)\right)$.

Putting together the arguments above, we further obtain:

$$\mathbb{E}\left[f(\boldsymbol{\theta}') - f^* | \boldsymbol{\theta}\right] \leq (f(\boldsymbol{\theta}) - f^*)(1 + \frac{48k^*}{k-k^*} \cdot \frac{\beta}{\alpha} - \frac{\beta\eta^2\alpha}{2}) + 2\beta\eta^2\sigma^2\frac{k+2k^*}{k-k^*}$$

Setting $k = k^* \cdot \left(768\left(\frac{\beta}{\alpha}\right)^2 + 1\right)$ we have $\frac{48k^*}{k-k^*} \cdot \frac{\beta}{\alpha} \leq 48 \cdot \frac{1}{768 \cdot (\beta/\alpha)^2} \cdot \frac{\beta}{\alpha} = \frac{1}{16} \cdot \frac{\alpha}{\beta}$. Moreover, we have $\frac{3k^*}{k-k^*} = \frac{3}{768} \cdot (\frac{\alpha}{\beta})^2 < \frac{1}{200}$, and so we can bound $2\beta\eta^2\sigma^2\frac{2k^*+k}{k-k^*} \leq \frac{401\sigma^2}{800\beta} < \frac{\sigma^2}{\beta}$. Therefore,

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}'\right) - f^* | \boldsymbol{\theta}\right] \leq \left(f\left(\boldsymbol{\theta}\right) - f^*\right)\left(1 - \frac{\alpha}{16\beta}\right) + \frac{\sigma^2}{\beta} \ .$$

Taking expectation over the entire history, this shows that after T steps, we have:

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_T) - f^*] &\leq (f(\boldsymbol{\theta}_0) - f^*)\left(1 - \frac{\alpha}{16\beta}\right)^T + \frac{\sigma^2}{\beta} \cdot \frac{1 - (1 - \frac{\alpha}{16\beta})^T}{1 - (1 - \frac{\alpha}{16\beta})} \\ &\leq (f(\boldsymbol{\theta}_0) - f^*)\left(1 - \frac{\alpha}{16\beta}\right)^T + \frac{16\sigma^2}{\alpha}\end{aligned}$$

Setting a desired error $\epsilon > 0$, such that $\epsilon = (f(\boldsymbol{\theta}_0) - f^*)\left(1 - \frac{\alpha}{16\beta}\right)^T$, we obtain that $T = O\left(\left(\frac{\beta}{\alpha}\right)\ln\frac{f(\theta_0)-f^*}{\epsilon}\right)$. Here, we have used the approximation $\ln(1-x) \approx -x$, for small x. Therefore, under the assumptions of the theorem, when performing stochastic IHT for $T = O\left(\left(\frac{\beta}{\alpha}\right)\ln\frac{f(\theta_0)-f^*}{\epsilon}\right)$ iterations, we obtain an iterate $\boldsymbol{\theta}_T$ such that

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_T\right) - f^*\right] \leq \epsilon + \frac{16\sigma^2}{\alpha} \ ,$$

which concludes the proof. □

A.1.3 Theoretical Guarantees for AC/DC

In this section we provide the proof for Corollary 3.3.1, which gives us theoretical guarantees for the practical AC/DC algorithm. The proof is based on a similar analysis used for Theorem 3.3.1.

First, we restate the Corollary, for completeness.

Corollary 3.3.1 (Convergence of AC/DC)**.** *Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function that decomposes as $f(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^m f_i(\boldsymbol{\theta})$, and has a k^*-sparse minimizer $\boldsymbol{\theta}^*$. Let $\beta > \alpha > 0$ be parameters, let $k = C \cdot k^* \cdot (\beta/\alpha)^2$ for some appropriately chosen constant C, suppose that each f_i is (N, β)-smooth, and L-Lipschitz, and that f is (k^*, α)-CPL.*

Let Δ_c and B be integers, and let $\{D_1, \ldots, D_B\}$ be a partition of $[m]$ into B subsets of cardinality $O(m/B)$ each. Given $\boldsymbol{\theta}$, let $g_{\boldsymbol{\theta}}^{(i)} = \frac{1}{|D_i|}\sum_{j\in D_i}\nabla f_j(\boldsymbol{\theta})$.

Suppose we replace the IHT iteration with a dense/sparse phase consisting of

1. *Δ_c dense phases during each of which we perform a full pass over the data and update the parameters through the iteration $\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta g_{\boldsymbol{\theta}}^{(i)}$ for all $i \in [B]$, with an appropriate step size η;*

2. *a pruning step which applies the Top-K operator T_k over the weights θ;*

3. *an optional sparse training phase which fully optimizes f over the sparse support given by the Top-K operator*

For initial parameters $\boldsymbol{\theta}_0$ and precision $\epsilon > 0$, this algorithm converges in $O\left(\frac{\beta}{\alpha} \cdot \ln \frac{f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*)}{\epsilon}\right)$ dense/sparse phases to a point $\boldsymbol{\theta}$ with $\|\boldsymbol{\theta}\|_0 \leq k$, such that

$$f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*) \leq \epsilon + O\left(\frac{L^2}{\alpha}\right).$$

Proof of Corollary 3.3.1. We will show that the proof mainly follows from that of Theorem 3.3.1. The main challenge is handling the error introduced by performing Δ_c stochastic gradient passes through the data, instead of a single stochastic gradient step. We first introduce a few notations. First, let $M = \Delta_c \cdot B$ and assume that compared to a single stochastic gradient step for which we used a learning rate η, here we use a dampened learning rate $\eta' = \frac{\eta}{M}$. Let $\boldsymbol{\theta}$ be the current iterate with support S, and we define the following sequence of stochastic gradient steps, before applying the Top-K operator:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta' \cdot g_{\boldsymbol{x}_t},$$

where $0 \leq t \leq M-1$, $\boldsymbol{x}_0 = \boldsymbol{\theta}$, and each $g_{\boldsymbol{x}_t}$ is a stochastic gradient computed on a mini-batch of samples of size $\frac{m}{B}$, such that for each consecutive B steps, the corresponding gradients cover the entire training set.

It is easy to see that after M steps, we have $\boldsymbol{x}_M = \boldsymbol{\theta} - \eta' \cdot \sum_{t=0}^{M-1} g_{\boldsymbol{x}_t}$. We denote the sum of stochastic gradients starting at $\boldsymbol{\theta}$ as $G_{\boldsymbol{\theta}} := \sum_{t=0}^{M-1} g_{\boldsymbol{x}_t}$. Then, our gradient update prior to applying the Top-K operator is $\widetilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \eta' \cdot G_{\boldsymbol{\theta}}$.

Since each f_i is β-smooth and L-Lipschitz, the same holds also for f, and for any mini-batch $\frac{1}{|D|}\sum_{i=1}^{D} f_i$. In particular, for each stochastic gradient $g_{\boldsymbol{x}_t}$, we have $\|g_{\boldsymbol{x}_t}\| \leq L$. Therefore,

$$\left\|\widetilde{\boldsymbol{\theta}} - \boldsymbol{\theta}\right\| = \eta' \cdot \left\|\sum_{t=0}^{M-1} g_{\boldsymbol{x}_t}\right\| \leq \eta' \cdot \sum_{t=0}^{M-1} \|g_{x_t}\| \leq \eta' M L = \eta L$$

Based on this, we can also bound the distance to $\boldsymbol{\theta}$ from any intermediate update $\boldsymbol{x}_t$ as $\|\boldsymbol{x}_t - \boldsymbol{\theta}\| \leq \eta L$. Moreover, since each f_i is smooth, we have that for any vectors $\boldsymbol{x}, \boldsymbol{y}$, $\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\| \leq \beta \|\boldsymbol{x} - \boldsymbol{y}\|$; thus, for intermediate updates $\boldsymbol{x}_t$, we have $\|\nabla f_i(\boldsymbol{x}_t) - \nabla f_i(\boldsymbol{\theta})\| \leq \beta\eta L$.

Therefore, putting everything together, we can bound:

$$\|\eta' G_{\boldsymbol{\theta}} - \eta \nabla f(\boldsymbol{\theta})\| = \eta \left\|\frac{1}{M}\sum_{t=0}^{M-1} g_{\boldsymbol{x}_t} - \nabla f(\boldsymbol{\theta})\right\| \leq \eta^2 \beta L \qquad (**)$$

For the last inequality, we used the fact that every B consecutive intermediate steps cover the entire training set, which means we can directly map the f_i components of each sub-sequence of $g_{\boldsymbol{x}_t}$ gradients (with $Bu + 1 \leq t \leq B(u+1)$, for $1 \leq u \leq \Delta_c$) to the corresponding components from $\nabla f(\boldsymbol{\theta})$.

Given this last inequality, we can now show how the result follows. We first denote our stochastic IHT update $\boldsymbol{\theta}' = T_k(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}})$. Note that since f is (N, β)-smoooth, it is also automatically

smooth along sparse vectors. Thus, we use Lemma A.1.2 for $\delta = T_k(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}}) - \boldsymbol{\theta}$, restricted to the support $S \cup S'$:

$$f(\boldsymbol{\theta}') \leq f(\boldsymbol{\theta}) + \frac{\beta}{2}\left\|\left(\frac{1}{\beta}\nabla f(\boldsymbol{\theta}) + T_k(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}}) - \boldsymbol{\theta}\right)_{S\cup S'}\right\|^2 - \frac{1}{2\beta}\left\|\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2$$
$$\leq f(\boldsymbol{\theta}) + \beta\left\|(T_k(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}}) - (\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}}))_{S\cup S'}\right\|^2 + \beta\left\|\eta' G_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f(\boldsymbol{\theta})\right\|^2 - \frac{1}{2\beta}\left\|\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2$$
$$= f(\boldsymbol{\theta}) + \beta\left\|(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}})_{S\setminus S'}\right\|^2 + \beta\left\|\eta' G_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f(\boldsymbol{\theta})\right\|^2 - \frac{1}{2\beta}\left\|\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2$$

Now we set $\eta = \frac{1}{2\beta}$ and have the following bound on $\left\|(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}})_{S\setminus S'}\right\|^2$:

$$\beta\left\|(\boldsymbol{\theta} - \eta' G_{\boldsymbol{\theta}})_{S\setminus S'}\right\|^2 \leq 2\beta\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 + 2\beta\left\|(\eta' G_{\boldsymbol{\theta}} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2$$
$$\leq \frac{1}{\eta}\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 + \frac{L^2}{8\beta},$$

where for the last inequality we used (**) and the fact that $\eta = \frac{1}{2\beta}$.

Then,

$$f(\boldsymbol{\theta}') \leq f(\boldsymbol{\theta}) + \frac{1}{\eta}\left(\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2\right) + \beta\left\|\eta' G_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f(\boldsymbol{\theta})\right\|^2 + \frac{L^2}{8\beta}$$

We can further bound the term

$$\beta\left\|\eta' G_{\boldsymbol{\theta}} - \frac{1}{\beta}\nabla f(\boldsymbol{\theta})\right\|^2 \leq 2\beta\left\|\eta' G_{\boldsymbol{\theta}} - \eta\nabla f(\boldsymbol{\theta})\right\|^2 + 2\beta\left\|\eta\nabla f(\boldsymbol{\theta})\right\|^2 \leq \frac{L^2}{8\beta} + \frac{L^2}{2\beta},$$

where we used again (**) and the L-Lipschitz property of f.

Therefore,

$$f(\boldsymbol{\theta}') \leq f(\boldsymbol{\theta}) + \frac{1}{\eta}\left(\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2\right) + \frac{3L^2}{4\beta}$$

Now, similar to the proof from Theorem 3.3.1, we can apply Corollary A.1.1 to upper-bound $\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2$, after defining $S^* = \text{supp}(T_{k^*}(\nabla f(\boldsymbol{\theta})))$:

$$\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2 \leq \frac{3k^*}{k-k^*}\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_T - \boldsymbol{\theta}^*\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S^*}\right\|^2,$$

where $|T| \leq 3k^* + k$ and $\text{supp}(\boldsymbol{\theta}^*) \subseteq T$. Then, we apply the same argument as for Theorem 3.3.1, using the CPL property and inequality (*) to further bound:

$$\left\|(\boldsymbol{\theta} - \eta\nabla f(\boldsymbol{\theta}))_{S\setminus S'}\right\|^2 - \left\|\eta\nabla f(\boldsymbol{\theta})_{S\cup S'}\right\|^2 \leq (f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*))(\frac{3k^*}{k-k^*}\cdot\frac{8}{\alpha} - \eta^2\frac{\alpha}{2})$$

Finally, putting everything together, we obtain:

$$f(\boldsymbol{\theta}') \leq f(\boldsymbol{\theta}) + (f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*))(\frac{48k^*}{k-k^*}\cdot\frac{\beta}{\alpha} - \frac{\alpha}{4\beta}) + \frac{3L^2}{4\beta}.$$

Setting $k = k^*(8 \cdot 48 \cdot \frac{\beta^2}{\alpha^2} + 1)$, we obtain:

$$f(\boldsymbol{\theta}') - f(\boldsymbol{\theta}^*) \leq (f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}^*))(1 - \frac{\alpha}{8\beta}) + \frac{3L^2}{4\beta}$$

Taking expectation over the entire history, if $\boldsymbol{\theta}_0$ is the initial point, we obtain after T iterations:

$$\mathbb{E}[f(\boldsymbol{\theta}_T) - f(\boldsymbol{\theta}^*)] \leq (f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*))(1 - \frac{\alpha}{8\beta})^T + O(\frac{L^2}{\alpha}).$$

We further note that the steps performed during the sparse training phases do not affect convergence as they can only improve error in function value, which is the main quantity that our analysis tracks.

Therefore, for error $\epsilon > 0$, the algorithm after $T = O\left(\frac{\beta}{\alpha} \ln \frac{f(\boldsymbol{\theta}_0) - f(\boldsymbol{\theta}^*)}{\epsilon}\right)$ iterations reaches a point $\boldsymbol{\theta}_T$ such that

$$\mathbb{E}[f(\boldsymbol{\theta}_T) - f(\boldsymbol{\theta}^*)] \leq \epsilon + O(\frac{L^2}{\alpha}),$$

which concludes the proof.

□

A.2 Additional Experiments

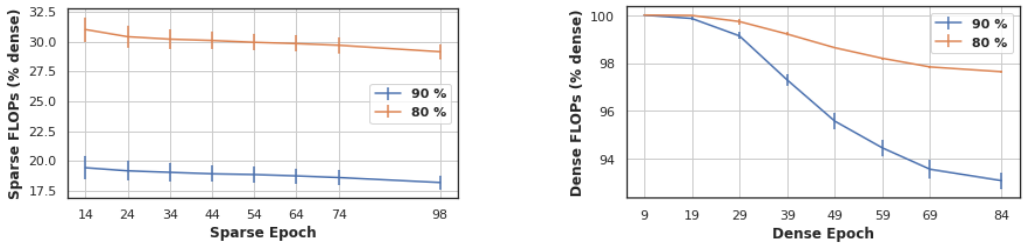
In this section we provide additional experimental details and comparisons between AC/DC and other pruning methods. We discuss additional properties of AC/DC on ImageNet models, a comparison with sparse training method Top-KAST [JPR+20], together with real-time inference speed-ups for sparse AC/DC models in Section A.2.1. Finally, we give additional details on the memorization experiments for AC/DC on CIFAR-10 in Section A.2.2

A.2.1 Additional Experiments on ImageNet

Additional Properties and Hyperparameter Details

Training Hyper-parameters for ImageNet. We used the same hyper-parameters for all our ImageNet experiments, on both ResNet50 and MobileNetV1. Namely, we trained using SGD with momentum and batch size 256. We used a cosine learning rate scheduler, after an initial warm-up phase of 5 epochs, when the learning rate was linearly increased to 0.256. The momentum value was 0.875 and weight decay was approximately 0.00003. These hyper-parameters have the standard values used in the implementation of STR [KRS+20].

Dynamics of FLOPs during training. Despite the dynamics of the compression masks presented in Figure 3.2b, we noticed that the sparsity distribution does not change substantially. This can be observed when looking at the number of inference FLOPs per sample, at the end of each compression phase, in Figure A.1a. Interestingly, as training progresses, AC/DC also induces structured sparsity, as more neurons and convolutional filters get pruned. This can be deduced from the decreasing inference FLOPs at the end of each dense phase, as shown in Figure A.1b. Interestingly, as previously described in Section 3.4.2, the resulting AC/DC dense models have a small percentage of zero-valued weights, most likely due to neurons or filters that are completely pruned away during the compression phase. We show the percentages of



(a) Test FLOPs after each sparse phase

(b) Test FLOPs after each dense phase

Figure A.1: Dynamics of sparse and dense inference FLOPs for ImageNet on ResNet50, as a percentage of the dense baseline FLOPs

zero weights in AC/DC dense models in Table A.1, where especially at high sparsity ($\geq 95\%$), a non-trivial percentage of the weights remain inactive; this in turn decreases the total FLOPs for the dense models.

AC/DC with uniform pruning. As discussed, for example, in [SA20], gradual magnitude pruning (GMP) with a global sparsity distribution tends to outperform its uniform counterpart. Interestingly, with global magnitude pruning later layers (which also tend to be the largest) are pruned the most. Moreover, we did not encounter convergence issues caused by entire layers being pruned, as hypothesized in some previous work [EGM+20, JPR+20]. However, one concern related to global magnitude pruning is a potential FLOP inefficiency of the resulting models; in theory, this would be a consequence of the earlier layers being pruned the least. For this reason, we performed additional experiments with AC/DC at uniform sparsity, with the first and last layers dense (as commonly used in the literature [EGM+20, JPR+20]). Our results show that there are no substantial differences compared to AC/DC with global magnitude pruning. However, keeping the first and last layers dense substantially improves the results with global magnitude pruning. These observations emphasize that AC/DC is an easy-to-use method which works reliably well with different pruning criteria. For complete results, please see Table A.2.

Target Sparsity	Top-1 Accuracy (%)	Inference FLOPs	Inactive Weights (%)
80	73.8	0.98×	3.2
90	73.1	0.93×	10.5
95	72.9	0.85×	22.0
98	70.8	0.67×	49.8

Table A.1: (ImageNet/ResNet50) Accuracy, sparsity, inference FLOPs and percentage of inactive weights for the resulting AC/DC dense models (before fine-tuning, one seed).

Sparsity Distribution	Target Sparsity(%)	Global Sparsity(%)	Top-1 Accuracy (%)	FLOPs Inference
global	90	89.8	75.14	0.18×
global*	90	82.6	75.64	0.21×
uniform*	90	82.6	75.04	0.13×
global	95	94.8	73.15	0.11×
global*	95	87.2	74.16	0.13×
uniform*	95	87.2	73.28	0.08×

Table A.2: (ImageNet/ResNet) AC/DC with uniform vs global magnitude pruning on ResNet50 (one seed), where ($\star$) denotes that the first and last layers are dense.

Length of compression/decompression phases. It is important to note that the length of the dense and sparse phases used in AC/DC, together with the warm-up and fine-tuning phases, could have a significant impact on the quality of the resulting models. Before settling on the sparsity pattern we used for all our ImageNet experiments (see Figure 3.2a, we experimented with different lengths for the sparse/dense phases, but found that ultimately the pattern used in the paper had the best trade-off between training FLOPs and validation accuracy. Due to computational limitations, and to ensure a fair comparison with the dense baseline and

Model/Setup	Real-Time Inference, 4 cores	Batch 64 Inference, 16 cores
ResNet50 ONNXRT v1.6	14.773	329.734
ResNet50 Dense	15.081	285.958
ResNet50 90% Pruned	9.46	124.193
ResNet50 90% Unif. Pruned	8.495	116.897
MobileNetV1 ONNXRT v1.6	2.552	80.748
MobileNetV1 Dense	2.513	55.845
MobileNetV1 Pruned 75%	1.96	40.976
MobileNetV1 Pruned 90%	1.468	34.909

Table A.3: Practical inference speed-up achieved with AC/DC models. The table shows the time per batch (milliseconds) using a sparse inference engine [Dee21].

other pruning methods, we decided on using a fixed number of 100 training epochs (the same used for the dense baseline). In this setup, we experimented mainly with the lengths for the compression/decompression phases used in Figure 3.2a, but noticed that having a longer final decompression phase had a positive impact on the fine-tuned dense model. For instance, when following a sparsity schedule as in Figure 3.1 (i.e. a shorter final decompression phase of 5 epochs and last 10 epochs of sparse finetuning), the sparse model at 90% sparsity had a similar performance to the reported results (75.18% accuracy, from one seed), while the fine-tuned dense model was substantially below the dense baseline (76.05% validation accuracy). We believe having a short warm-up period and a longer fine-tuning phase are both beneficial for the sparse model; in our experiments, we only used warm-up phases of 10 epochs, but believe that shorter phases are worth exploring as well. Furthermore, the mask difference between consecutive compression phases is an important guide for choosing the sparsity schedule: as it was previously discussed, having a non-trivial difference between the masks typically results in better sparse models.

Inference Speedups

We examine the potential for real-world speedup of models produced through our framework. For this, we use the CPU-based inference framework of [Dee21], which supports efficient inference over unstructured sparse models, and is free to use for non-commercial purposes. Specifically, we export our PyTorch-trained models to the ONNX intermediate format, preserving weight sparsity, and then execute inference on a subset of samples, at various batch sizes, measuring time per batch. We perform our tests on an Intel i9-7980XE CPU with 16 cores and 2.60GHz core frequency and simulate two scenarios: the first is *real-time inference*, i.e. samples are processed one at a time, in a resource-constrained environment, using only 4 cores. The second is *batch inference*, for which we pick batch size 64, in a cloud environment, for which we use all 16 cores. We measure average time per batch for the sparse models against dense baselines, for which we use both the Deepsparse engine, and the ONNX runtime (ONNXRT). We present the average over 10 runs. The variance is extremely low, so we omit it for readability.

We now briefly discuss the results. First, notice that the dense baselines offer similar performance for real-time inference, but that the Deepsparse engine has a slight edge at batch 64. We will therefore compare against its timings below. The results show a speedup of 1.6x for the 90% global-pruned ResNet50 model, and 1.8x for the uniformly pruned one: the uniformly-pruned model is slightly faster, which correlates with its lower FLOP count. This pattern is preserved in MobileNetV1 experiments, although the speedups are relatively lower, since the architecture

Method	80% Sparsity		90% Sparsity	
	Backward (%)	Accuracy (%)	Backward (%)	Accuracy (%)
AC/DC	80 / 0	**76.3** ± 0.1	90 / 0	**75.03** ± 0.1
Top-KAST	0	75.64	0	74.42
Top-KAST	50	74.78	50	74.09
Top-KAST	80	72.19	80	73.07

Table A.4: (ImageNet/ResNet50) Comparison with Top-KAST, with different sparsity values used in the backward pass, when pruning all layers.

is more compact. We note that the speedups are more substantial when performing inference over a batch, where the engine has more potential for parallelization, and our setup uses more cores.

Comparison Between AC/DC and Sparse Training Methods

As previously highlighted, Top-KAST is the closest to us, in terms of validation accuracy, out of existing sparse training methods. However, for the results reported, the authors kept the first convolutional and final fully-connected layers dense. To obtain a fair comparison, we used AC/DC on the same sparse distribution, and for 90% sparsity over the pruned layers (82.57% overall network sparsity), our results improved substantially. Namely, the best sparse model reached 75.64% validation accuracy (0.6% increase from the results in Table A.2), while the accuracy of the best dense model was 76.85% after fine-tuning. For completeness, we also provide in Table A.4 the results for Top-KAST when all layers are pruned, as they were provided to us by the authors. Notice that AC/DC surpasses even Top-KAST with dense back-propagation.

It is important to note, however, that because of its flexibility in choosing the gradients density, Top-KAST can theoretically obtain substantially better training speed-ups than AC/DC, the latter being constrained by its dense training phases. This allows Top-KAST to improve the accuracy of the models by increasing the number of training epochs, while still enabling (theoretical) training speed-up. We present in Table A.5 another comparison between AC/DC and Top-KAST, when the training time for the latter is increased 2 or 5 times. For all results (which were provided to us by the authors), the first and last layers for Top-KAST are dense. When comparing with AC/DC with all layers pruned, Top-KAST obtains better results at 98% and 95% sparsity, with increased training epochs. However, when the first and last layers are kept dense, the results for AC/DC at 95% and 98% sparsity are better than for Top-KAST with increased training steps. For all the results reported on AC/DC the number of training steps was fixed at 100 epochs.

We note that the results obtained with AC/DC can be improved as well with increased number of training epochs. As an example, when using the same sparsity schedule extended over 150 epochs, the best sparse model obtained with AC/DC on 90% sparsity reached $\approx 76\%$ accuracy, using fewer training FLOPs compared to the original dense baseline trained on 100 epochs (namely 87%). Furthermore, when we fine-tune the dense model by replacing the final 15 epochs compression phase with dense training, we obtain a dense model with 76.95% accuracy, slightly higher than the original dense baseline.

Method	Sparsity (%)	Backward Sparsity (%)	Sparse Top-1 Accuracy (%)	Train FLOPs (%)	Inference FLOPs (%)
AC/DC	80	80 / 0	**76.3** ± 0.1	$0.65\times$	$0.29\times$
Top-KAST	80	0	75.59	$0.48\times$	$0.23\times$
Top-KAST	80	60	74.59	$0.29\times$	$0.23\times$
Top-KAST 2x	80	0	76.11	$0.97\times$	$0.23\times$
Top-KAST 2x	80	60	75.29	$0.58\times$	$0.23\times$
AC/DC	90	90 / 0	75.03 ± 0.1	$0.58\times$	$0.18\times$
AC/DC*	90	90 / 0	**75.64**	$0.6\times$	$0.21\times$
AC/DC* (unif.)	90	90/0	75.04	$0.55\times$	$0.13\times$
Top-KAST	90	0	74.65	$0.42\times$	$0.13\times$
Top-KAST	90	80	73.03	$0.16\times$	$0.13\times$
Top-KAST 2x	90	0	75.35	$0.84\times$	$0.13\times$
Top-KAST 2x	90	80	74.16	$0.32\times$	$0.13\times$
AC/DC	95	95 / 0	73.14 ± 0.2	$0.53\times$	$0.11\times$
AC/DC*	95	95 / 0	**74.16**	$0.54\times$	$0.13\times$
AC/DC* (unif)	95	95 / 0	73.28	$0.5\times$	$0.08\times$
Top-KAST	95	0	71.83	$0.39\times$	$0.08\times$
Top-KAST	95	90	70.42	$0.1\times$	$0.08\times$
Top-KAST 2x	95	0	73.29	$0.77\times$	$0.08\times$
Top-KAST 2x	95	90	72.42	$0.19\times$	$0.08\times$
Top-KAST 5x	95	0	**74.27**	$1.94\times$	$0.08\times$
Top-KAST 5x	95	90	73.17	$0.48\times$	$0.08\times$
AC/DC	98	98 / 0	68.44 ± 0.09	$0.46\times$	$0.06\times$
AC/DC*	98	98 / 0	**71.27**	$0.47\times$	$0.08\times$
Top-KAST	98	90	67.06	$0.08\times$	$0.05\times$
Top-KAST	98	95	66.46	$0.06\times$	$0.05\times$
Top-KAST 2x	98	90	68.99	$0.15\times$	$0.05\times$
Top-KAST 2x	98	85	68.87	$0.12\times$	$0.05\times$

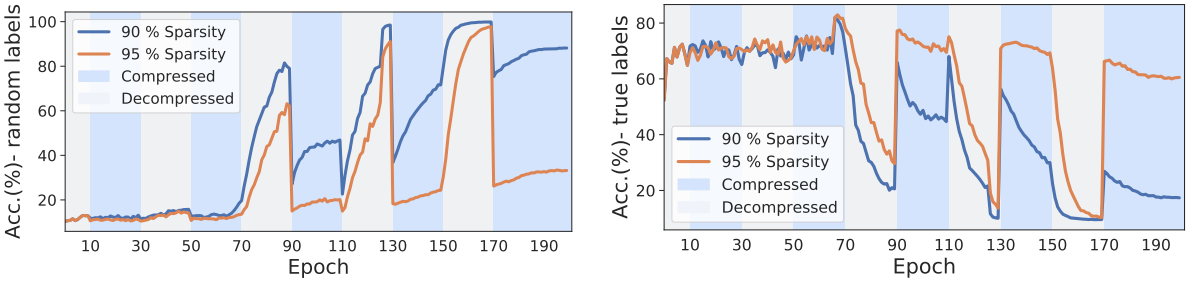
Table A.5: Comparison with Top-KAST with increased training steps (ResNet50). ($\star$) indicates that the first and last layers are dense for AC/DC, while this is the case for all Top-KAST results.

A.2.2 Memorization Experiments on CIFAR-10

In what follows, we study the similarities between the sparse and dense models learned with AC/DC, on the particular setup of memorizing random labels. Specifically, we select 1000 i.i.d. training samples from the CIFAR-10 dataset [KH$^+$09] and randomly change their labels. We train a ResNet20 [HZRS16] model using AC/DC, at various target sparsity levels, ranging from 50% to 95%. We use SGD with momentum, weight decay, and initial learning rate 0.1 which is decayed by a factor of 10 every 60 epochs, starting with epoch 65.

Using data augmentation dramatically affects the memorization of randomly-labelled training samples, and thus we differentiate between the two possible cases. Namely, the regular baseline can easily memorize (in the sense of reaching perfect accuracy) the randomly-labelled samples, when *no* data augmentation is used; in comparison, with data augmentation memorization is more difficult, and the accuracy on randomly-labelled samples for the baseline is just above 60%. In addition to the accuracy on the perturbed samples with respect to their new random labels, we also track the accuracy with respect to the "true" or correct labels. This differentiation offers a better understanding regarding where memorization fails and a glimpse into the robustness properties of neural networks in general, and of AC/DC, in particular.

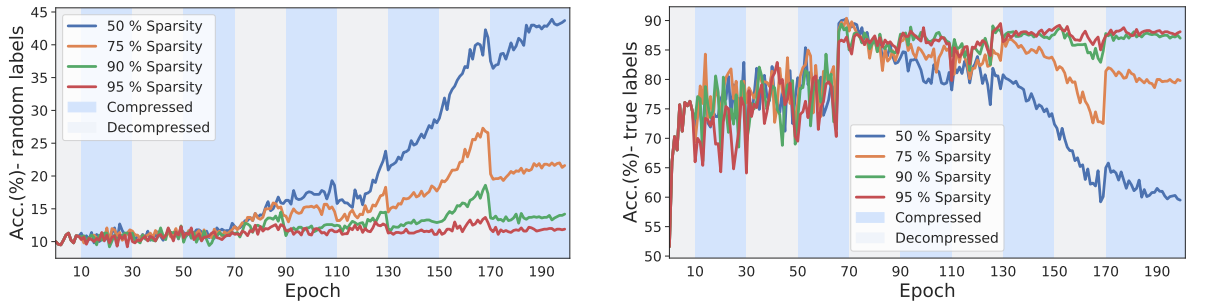
No data augmentation. As previously mentioned, in this case the baseline model can perfectly memorize the perturbed data, with respect to their random labels. Interestingly, prior



(a) Accuracy on the mis-labelled data

(b) Accuracy on the mis-labelled data (w.r.t. the true labels)

Figure A.2: Accuracy during training with AC/DC at 90% and 95% target sparsity, for 1000 randomly labelled CIFAR-10 images. No data augmentation was applied to the training samples.



(a) Accuracy on the mis-labelled data

(b) Accuracy on the mis-labelled data (w.r.t. the true labels)

Figure A.3: Accuracy during training with AC/DC at 50%, 75%, 90% and 95% target sparsity, for 1000 randomly labelled CIFAR-10 images. Here, all samples were trained using data augmentation.

to the initial learning rate decay, most ($\geq 70\%$) perturbed samples are still correctly classified with respect to their "true" labels, and memorization happens very quickly after the learning rate is decreased. In the case of AC/DC with low target sparsity (50% and 75%), memorization has a very similar behavior to the dense baseline. However, for higher sparsity levels (90% and 95%) we can see a clear difference between the sparse and dense models. Namely, during each compression phase most perturbed samples are correctly classified with respect to their true labels, whereas in decompression phases their random labels are memorized. This phenomenon is illustrated in Figure A.2.

Data augmentation. In this case, memorization of the perturbed samples is more difficult, and it happens later on during training, usually after the second learning rate decrease for the baseline model. Interestingly, in the case of AC/DC we can see (Figure A.3) a clear inverse relationship between the amount of memorization and the target sparsity. Although low sparsity enables more memorization, most perturbed samples are still correctly classified with respect to their true labels. For higher sparsity levels (90% and 95%), most perturbed samples are correctly classified with respect to their true labels (almost 90%) and very few are memorized. Furthermore, the dense model resulted from AC/DC training is more robust than the original baseline, as it still learns the correct labels of the perturbed samples, despite being presented with random ones.

A.2.3 Computational Details

Hardware Details

Experiments were run on NVIDIA RTX 2080 GPUs for image classification tasks, and NVIDIA RTX 3090 GPUs for language modelling. Each ImageNet run took approximately 2 days for ResNet50 and one day for MobileNet, while each Transformer-XL experiment took approximately 2 days.

FLOPs Computation

When computing FLOPs, we take into account the number of zero-valued weights for linear and convolutional layers. To compute the FLOPs required for a backward pass over a sample, we use the same convention as RigL [EGM$^+$20]; namely, if F denotes the inference FLOPs per sample, the number of backward FLOPs is estimated as $B = 2 \cdot F$, as we need F FLOPs to backpropagate the error, and additional F to compute the gradients w.r.t. the weights. For ImageNet experiments, we ignore the FLOPs required for Batch Normalization, pooling, ReLU or Cross Entropy, similarly to other methods [EGM$^+$20, SA20, KRS$^+$20]; however, these layers have a negligible impact on the total FLOPs number (at most $0.01\times$ the dense number).

For compression and decompression phases C and D, we consider F_C and F_D the compression and decompression inference FLOPs per sample, respectively. We use F to denote the inference FLOPs per sample for the baseline network. During each compression phase, the training FLOPs per sample can be estimated as $3 \cdot F_C$. For decompression phases, we noticed that a small fraction of weights remain zero, and therefore $F_D < F$. When doing a backward pass we have additional F_D from back-propagating the error, and F extra FLOPs for the gradients with respect to all parameters. Therefore, we estimate the training FLOPs per sample during a decompression phase as $2 \cdot F_D + F$. We measure the number of FLOPs on a random input sample, at the end of each training epoch and use this value to estimate the total training FLOPs for that particular epoch. To obtain the final number of FLOPs, we compute the inference FLOPs on a random input sample, estimate the backward FLOPs, compute the estimated training FLOPs over all training epochs as described above, and scale by the number of training samples.

APPENDIX B

Appendix for Chapter 4

B.1 Hyperparameters and Training Setup

Here we discuss the general hyperparameters and experimental setup used for the full and linear finetuning experiments. Regarding data loading image augmentation settings, we are careful to match them to the ones used in the original upstream training protocol. Specifically, this affects the choice of whether to use Bicubic or Bilinear image interpolation for image resizing; for example, RigL models were trained using Bicubic interpolation, whereas the other pruning methods considered used the Biliniar interpolation. All ResNet and MobileNet models considered were trained using standard ImageNet-specific values for the normalization mean and standard deviation. In the case of full finetuning, we used dataset-specific normalization values for the downstream tasks; these were obtained by loading the dataset once with standard data augmentations and computing the means and variances of the resulting data. For linear finetuning, we use center cropping of the images, followed by normalization using standard ImageNet values. For both full and linear finetuning, we use the same training hyperparameters as [SIE+20]; specifically, we train for 150 epochs, decreasing the initial learning rate by a factor of 10 every 50 epochs. We use 0.01 as the initial learning rate for all linear finetuning experiments; for full finetuning, we empirically found 0.001 to be the initial learning rate which gives comparable results for most datasets except Aircraft and Cars, for which we use 0.01. Our experiments were conducted using PyTorch 1.8.1 and NVIDIA GPUs. All full finetuning experiments on the ResNet50 backbone were repeated three times and all linear finetuning experiments five times.

B.2 Linear and Full Finetuning Results on ResNet50

In this section, we provide additional details, together with the complete results for our experiments for linear and full finetuning from ResNet50, presented in Sections 4.4.2 and 4.4.3. For each pruning method, we used a range of sparsity levels, and trained linear and full finetuning for each model and sparsity level, on all 12 downstream tasks; each experiment was repeated 5 times for linear and 3 times for full finetuning. Note that checkpoints for some pruning methods were not available for some of the higher sparsities.

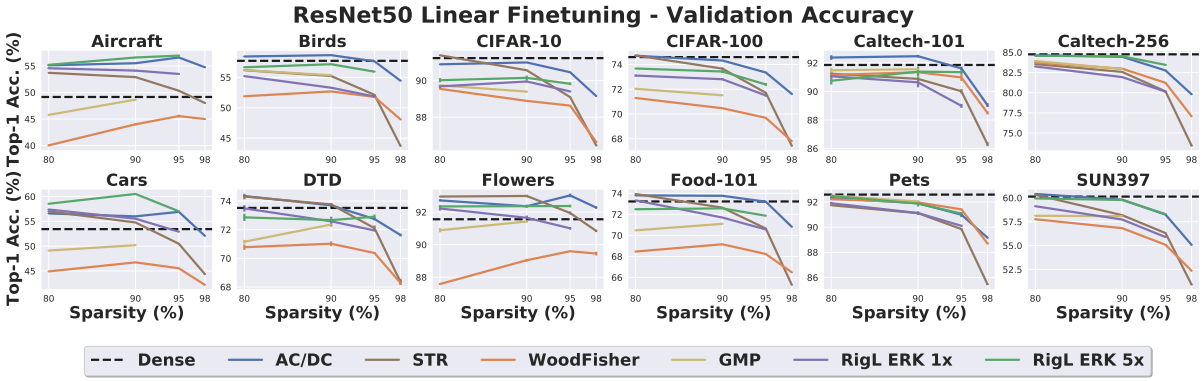


Figure B.1: (ResNet50/Linear Finetuning) Per-dataset downstream validation accuracy.

B.2.1 Linear Finetuning Results

We provide the complete results for our linear finetuning experiments on each downstream task, for all pruning methods and sparsity levels considered. The results for the transfer accuracies for each pruning strategy, sparsity level, and downstream task are presented in Figure B.1 and Table B.2. We discussed in Section 4.4.2 that regularization methods match and even sometimes outperform the dense baseline transfer performance. Note that this fact is valid not only in aggregate, but also at the level of each individual dataset.

In Table B.2, we also include the linear transfer results for LTH-T. We note that the generally poor performance of the method, especially for more specialized tasks and higher sparsity levels, should not be taken as a criticism of the method itself: this use case is clearly contrary to the method's design, and the spirit of the original Lottery Ticket Hypothesis (which aims to discover masks with the intent to retrain, rather than final weights). Rather, we include these results to provide quantitative justification for the omission of LTH-T from any further analyses, and supporting the original authors' point that additional finetuning is *necessary* in order to obtain a competitive lottery ticket for transfer learning.

Results With Different Optimizers. Additionally, we validate our linear finetuning results by training with a different optimizer than SGD with momentum; namely, we use L-BFGS [LN89] and L_2 regularization. To select the L_2 regularization hyperparameter, we use 20% of the training samples in each downstream task for validation, and perform linear finetuning on each task using multiple values of the L_2 regularization. After selecting the best value for L_2 on the validation set, we retrain on the entire training set, and report the final accuracy on the test set. Our hyperparameter selection strategy is consistent with the one used in [KSL19] We present the full results of linear finetuning using L-BFGS on the 12 downstream tasks in Table B.1. Generally, we observe that the test accuracies for linear finetuning with L-BFGS are similar or slightly better compared to the results obtained when using SGD. This could be a consequence of a better hyperparameter search, also tuned individually for each task; in contrast, for the SGD results, we used the same setup for all downstream tasks, based on the same hyperparameters used in [SIE+20]. Overall, these results confirm our initial findings that sparse regularization methods perform better on linear finetuning, compared to progressive sparsification methods.

Method	Dense	AC/DC	GMP	RigL ERK 1x	RigL ERK 5x	STR	WoodFisher
				80% Sparsity			
Aircraft	50.3	**56.5**	46.7	54.8	55.7	54.7	43.3
Birds	56.7	**58.2**	55.8	55.2	56.4	55.6	52.0
CIFAR-10	91.0	90.6	89.7	90.1	90.7	**91.2**	89.1
CIFAR-100	74.4	**74.6**	71.1	73.3	74.4	74.5	70.4
Caltech-101	91.6	**92.0**	90.5	90.1	91.0	90.6	91.1
Caltech-256	**84.4**	84.2	83.9	83.2	**84.4**	83.4	83.6
Cars	56.2	59.7	49.6	58.6	**60.2**	59.9	46.4
DTD	73.2	**74.5**	70.9	73.8	73.5	73.7	70.9
Flowers	93.0	93.5	92.0	93.1	93.4	**93.7**	88.4
Food-101	73.2	73.8	70.6	73.5	**74.0**	73.2	68.9
Pets	**92.2**	91.7	92.1	91.1	**92.2**	91.6	**92.2**
SUN397	60.2	60.4	59.4	60.1	**60.8**	60.6	58.8
				90% Sparsity			
Aircraft	50.3	56.1	49.1	55.4	**57.5**	54.7	44.9
Birds	56.7	**58.3**	55.2	53.0	57.3	54.7	52.7
CIFAR-10	**91.0**	90.9	89.5	90.3	90.7	90.2	88.8
CIFAR-100	**74.4**	74.2	71.3	72.9	**74.4**	73.5	69.6
Caltech-101	91.6	**91.9**	91.4	90.3	91.8	90.0	91.1
Caltech-256	84.4	84.1	82.6	81.9	**84.6**	82.2	82.9
Cars	56.2	57.9	52.4	57.1	**61.9**	57.6	48.2
DTD	**73.2**	72.0	70.9	72.9	72.1	72.2	70.8
Flowers	93.0	93.4	92.6	92.6	93.4	**93.9**	90.3
Food-101	73.2	73.6	71.3	72.2	**74.1**	72.4	69.4
Pets	**92.2**	91.1	91.8	91.2	91.7	90.7	91.5
SUN397	60.2	59.6	59.4	58.6	**60.7**	58.1	57.9
				95% Sparsity			
Aircraft	50.3	**57.4**	N/A	54.3	57.0	51.5	45.8
Birds	56.7	**57.0**	N/A	51.8	56.1	51.8	51.0
CIFAR-10	**91.0**	90.4	N/A	89.8	90.5	89.0	88.5
CIFAR-100	**74.4**	73.0	N/A	71.4	73.2	71.5	69.3
Caltech-101	91.6	90.6	N/A	89.2	**91.7**	89.8	90.2
Caltech-256	**84.4**	82.5	N/A	80.0	83.4	80.2	81.4
Cars	56.2	58.6	N/A	55.2	**58.9**	52.9	46.6
DTD	73.2	71.9	N/A	72.6	**73.5**	71.1	70.1
Flowers	93.0	**94.2**	N/A	92.2	93.4	92.8	90.5
Food-101	73.2	73.0	N/A	70.7	**73.4**	70.4	68.7
Pets	**92.2**	90.9	N/A	90.2	91.1	88.5	91.2
SUN397	**60.2**	58.1	N/A	56.7	59.2	56.7	56.5

Table B.1: (ResNet50/Linear Finetuning) Test accuracy when performing linear finetuning with the L-BFGS optimizer.

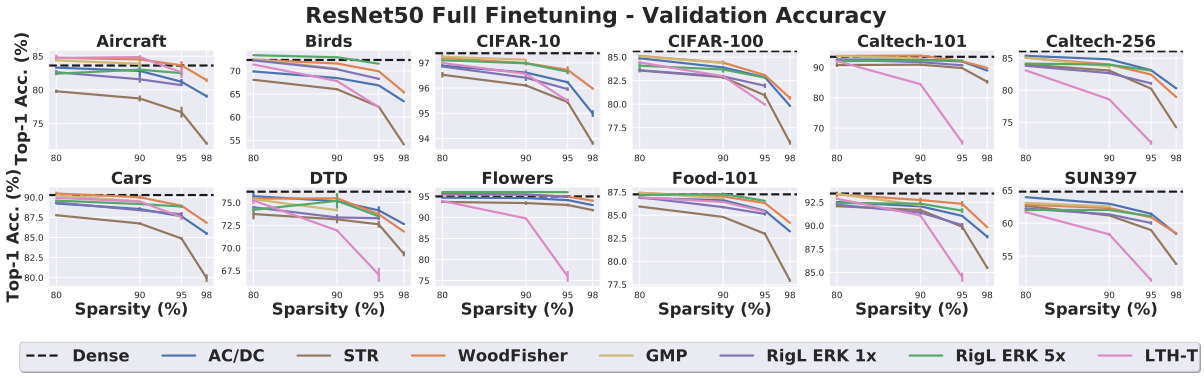


Figure B.2: (ResNet50/Full Finetuning) Per-dataset downstream validation accuracy.

B.2.2 Full Finetuning Results

Similarly to linear finetuning, we further provide complete results for full finetuning from sparse models. We present individual results per downstream task and pruning method, at different sparsity levels, in Figure B.2 and Table B.3; we report for each the average and standard deviation across 3 different trials. The results further support our conclusions from Section 4.4.3; namely, downstream task accuracy is correlated with the backbone sparsity,

and progressive sparsification methods (GMP, WoodFisher) generally perform better than regularization methods.

Model	Dense	AC/DC	GMP	LTH-T	RigL ERK 1x	RigL ERK 5x	STR	WoodFisher
				80% Sparsity				
Aircraft	49.2 ± 0.1	**55.1 ± 0.1**	45.8 ± 0.1	36.9 ± 0.1	54.6 ± 0.1	**55.2 ± 0.2**	53.7 ± 0.0	40.0 ± 0.2
Birds	57.7 ± 0.1	**58.4 ± 0.0**	56.2 ± 0.0	29.6 ± 0.1	55.2 ± 0.0	56.7 ± 0.1	56.2 ± 0.1	51.9 ± 0.1
CIFAR-10	91.2 ± 0.0	90.9 ± 0.0	89.7 ± 0.0	83.4 ± 0.1	89.7 ± 0.1	90.0 ± 0.1	**91.4 ± 0.0**	89.6 ± 0.0
CIFAR-100	**74.6 ± 0.1**	**74.7 ± 0.1**	72.0 ± 0.1	62.0 ± 0.1	73.1 ± 0.1	73.7 ± 0.0	**74.7 ± 0.0**	71.3 ± 0.0
Caltech-101	91.9 ± 0.1	**92.4 ± 0.2**	91.5 ± 0.2	75.4 ± 0.1	91.1 ± 0.1	90.8 ± 0.3	91.2 ± 0.1	91.2 ± 0.1
Caltech-256	**84.8 ± 0.1**	**84.6 ± 0.1**	83.9 ± 0.1	66.1 ± 0.1	83.3 ± 0.1	**84.6 ± 0.1**	83.6 ± 0.0	83.7 ± 0.1
Cars	53.4 ± 0.1	56.6 ± 0.0	49.1 ± 0.1	32.7 ± 0.1	57.4 ± 0.1	**58.6 ± 0.1**	57.0 ± 0.1	44.9 ± 0.1
DTD	73.5 ± 0.2	**74.4 ± 0.1**	71.2 ± 0.1	64.9 ± 0.2	73.5 ± 0.2	72.9 ± 0.3	**74.3 ± 0.2**	70.8 ± 0.2
Flowers	91.6 ± 0.1	92.7 ± 0.1	90.9 ± 0.1	85.6 ± 0.1	92.2 ± 0.1	92.3 ± 0.1	**93.0 ± 0.0**	87.6 ± 0.1
Food-101	73.2 ± 0.0	73.8 ± 0.0	70.5 ± 0.0	61.9 ± 0.0	73.3 ± 0.0	72.5 ± 0.1	**73.9 ± 0.0**	68.5 ± 0.0
Pets	**92.6 ± 0.1**	92.3 ± 0.1	**92.5 ± 0.1**	79.4 ± 0.1	91.9 ± 0.1	**92.5 ± 0.2**	91.7 ± 0.0	92.2 ± 0.1
SUN397	60.1 ± 0.0	**60.4 ± 0.0**	58.1 ± 0.0	47.4 ± 0.0	59.1 ± 0.1	59.9 ± 0.0	60.3 ± 0.0	57.8 ± 0.1
				90% Sparsity				
Aircraft	49.2 ± 0.1	55.5 ± 0.1	48.7 ± 0.1	16.5 ± 0.2	54.1 ± 0.1	**56.6 ± 0.1**	52.9 ± 0.1	44.0 ± 0.2
Birds	57.7 ± 0.1	**58.7 ± 0.0**	55.4 ± 0.1	11.4 ± 0.1	53.3 ± 0.0	57.2 ± 0.1	55.2 ± 0.1	52.7 ± 0.1
CIFAR-10	**91.2 ± 0.0**	91.0 ± 0.0	89.4 ± 0.0	67.0 ± 0.1	90.0 ± 0.1	90.2 ± 0.1	90.6 ± 0.0	88.9 ± 0.0
CIFAR-100	**74.6 ± 0.1**	74.3 ± 0.0	71.5 ± 0.0	42.2 ± 0.1	72.8 ± 0.1	73.4 ± 0.1	73.7 ± 0.1	70.5 ± 0.0
Caltech-101	91.9 ± 0.1	**92.5 ± 0.1**	91.6 ± 0.1	49.0 ± 0.6	90.6 ± 0.3	91.4 ± 0.4	90.9 ± 0.1	91.3 ± 0.1
Caltech-256	**84.8 ± 0.1**	84.5 ± 0.0	82.9 ± 0.0	42.0 ± 0.1	81.9 ± 0.0	84.5 ± 0.1	82.6 ± 0.0	83.0 ± 0.1
Cars	53.4 ± 0.1	56.0 ± 0.1	50.2 ± 0.0	15.4 ± 0.1	55.5 ± 0.1	**60.5 ± 0.1**	54.8 ± 0.1	46.7 ± 0.0
DTD	**73.5 ± 0.2**	**73.7 ± 0.2**	72.4 ± 0.2	54.7 ± 0.1	72.6 ± 0.3	72.7 ± 0.2	**73.8 ± 0.1**	71.0 ± 0.2
Flowers	91.6 ± 0.1	92.4 ± 0.0	91.4 ± 0.1	67.7 ± 0.1	91.6 ± 0.1	92.4 ± 0.1	**93.0 ± 0.1**	89.0 ± 0.1
Food-101	73.2 ± 0.0	**73.8 ± 0.0**	71.1 ± 0.0	46.9 ± 0.0	71.7 ± 0.0	72.6 ± 0.0	72.6 ± 0.0	69.2 ± 0.0
Pets	**92.6 ± 0.1**	91.9 ± 0.1	92.0 ± 0.1	43.8 ± 0.2	91.1 ± 0.1	91.9 ± 0.2	91.1 ± 0.1	92.0 ± 0.1
SUN397	**60.1 ± 0.0**	59.8 ± 0.1	58.1 ± 0.0	31.7 ± 0.1	57.7 ± 0.0	59.8 ± 0.1	58.2 ± 0.0	56.8 ± 0.0
				95% Sparsity				
Aircraft	49.2 ± 0.1	56.6 ± 0.1	N /A	4.5 ± 0.3	53.5 ± 0.1	**56.9 ± 0.1**	50.3 ± 0.1	45.6 ± 0.3
Birds	**57.7 ± 0.1**	**57.7 ± 0.0**	N /A	2.3 ± 0.1	51.9 ± 0.1	55.9 ± 0.0	52.1 ± 0.1	51.8 ± 0.1
CIFAR-10	**91.2 ± 0.0**	90.5 ± 0.0	N /A	39.9 ± 0.2	89.4 ± 0.0	89.8 ± 0.1	89.1 ± 0.0	88.6 ± 0.0
CIFAR-100	**74.6 ± 0.1**	73.4 ± 0.0	N /A	13.5 ± 0.2	71.5 ± 0.1	72.4 ± 0.1	71.7 ± 0.0	69.7 ± 0.0
Caltech-101	**91.9 ± 0.1**	91.6 ± 0.1	N /A	20.1 ± 0.5	89.0 ± 0.1	91.4 ± 0.1	90.0 ± 0.2	91.0 ± 0.2
Caltech-256	**84.8 ± 0.1**	82.8 ± 0.1	N /A	12.4 ± 0.3	80.1 ± 0.1	83.5 ± 0.1	80.2 ± 0.1	81.2 ± 0.1
Cars	53.4 ± 0.1	**56.9 ± 0.1**	N /A	3.9 ± 0.1	52.9 ± 0.0	**57.0 ± 0.1**	50.5 ± 0.1	45.5 ± 0.0
DTD	**73.5 ± 0.2**	72.7 ± 0.1	N /A	27.4 ± 0.2	71.9 ± 0.1	72.9 ± 0.2	72.1 ± 0.2	70.4 ± 0.1
Flowers	91.6 ± 0.1	**93.0 ± 0.1**	N /A	27.8 ± 0.6	91.0 ± 0.1	92.4 ± 0.1	91.9 ± 0.1	89.6 ± 0.0
Food-101	**73.2 ± 0.0**	**73.2 ± 0.0**	N /A	15.0 ± 0.1	70.6 ± 0.1	71.9 ± 0.0	70.7 ± 0.0	68.2 ± 0.0
Pets	**92.6 ± 0.1**	91.0 ± 0.2	N /A	15.9 ± 0.2	90.1 ± 0.1	91.1 ± 0.1	89.8 ± 0.1	91.4 ± 0.0
SUN397	**60.1 ± 0.0**	58.2 ± 0.0	N /A	8.4 ± 0.2	55.9 ± 0.1	58.3 ± 0.1	56.3 ± 0.0	55.1 ± 0.1
				98% Sparsity				
Aircraft	49.2 ± 0.1	**54.8 ± 0.1**	N /A	N /A	N /A	N /A	48.0 ± 0.1	45.0 ± 0.1
Birds	**57.7 ± 0.1**	54.5 ± 0.0	N /A	N /A	N /A	N /A	43.7 ± 0.0	48.1 ± 0.1
CIFAR-10	**91.2 ± 0.0**	89.2 ± 0.0	N /A	N /A	N /A	N /A	86.5 ± 0.0	86.6 ± 0.0
CIFAR-100	**74.6 ± 0.1**	71.6 ± 0.0	N /A	N /A	N /A	N /A	67.4 ± 0.0	67.8 ± 0.0
Caltech-101	**91.9 ± 0.1**	89.0 ± 0.1	N /A	N /A	N /A	N /A	86.3 ± 0.1	88.5 ± 0.1
Caltech-256	**84.8 ± 0.1**	79.8 ± 0.0	N /A	N /A	N /A	N /A	73.4 ± 0.1	77.1 ± 0.0
Cars	**53.4 ± 0.1**	52.1 ± 0.0	N /A	N /A	N /A	N /A	44.4 ± 0.1	42.2 ± 0.0
DTD	**73.5 ± 0.2**	71.6 ± 0.1	N /A	N /A	N /A	N /A	68.4 ± 0.2	68.3 ± 0.1
Flowers	91.6 ± 0.1	**92.3 ± 0.1**	N /A	N /A	N /A	N /A	90.8 ± 0.1	89.5 ± 0.1
Food-101	**73.2 ± 0.0**	70.8 ± 0.0	N /A	N /A	N /A	N /A	65.3 ± 0.0	66.5 ± 0.0
Pets	**92.6 ± 0.1**	89.2 ± 0.1	N /A	N /A	N /A	N /A	85.5 ± 0.1	88.7 ± 0.1
SUN397	**60.1 ± 0.0**	55.1 ± 0.0	N /A	N /A	N /A	N /A	50.9 ± 0.0	52.4 ± 0.0

Table B.2: (ResNet50) Full results showing transfer accuracy for sparse ResNet50 models with *linear finetuning*.

Model	Dense	AC/DC	GMP	LTH-T	RigL ERK 1x	RigL ERK 5x	STR	WoodFisher
				80% Sparsity				
Aircraft	83.6 ± 0.4	83.3 ± 0.1	**84.4 ± 0.2**	**84.7 ± 0.5**	82.6 ± 0.3	82.4 ± 0.2	79.8 ± 0.3	**84.8 ± 0.2**
Birds	72.4 ± 0.3	69.9 ± 0.2	72.5 ± 0.2	71.4 ± 0.1	72.3 ± 0.3	**73.4 ± 0.1**	68.1 ± 0.1	72.4 ± 0.4
CIFAR-10	**97.4 ± 0.0**	96.9 ± 0.1	97.2 ± 0.0	97.0 ± 0.0	96.9 ± 0.0	97.1 ± 0.0	96.5 ± 0.1	97.2 ± 0.1
CIFAR-100	**85.6 ± 0.2**	84.9 ± 0.2	85.1 ± 0.0	84.4 ± 0.2	83.6 ± 0.2	84.1 ± 0.4	83.6 ± 0.2	85.1 ± 0.1
Caltech-101	**93.5 ± 0.1**	92.5 ± 0.2	**93.7 ± 0.5**	92.1 ± 0.5	92.5 ± 0.1	92.0 ± 0.3	90.7 ± 0.6	**93.7 ± 0.1**
Caltech-256	**86.1 ± 0.1**	85.4 ± 0.2	85.1 ± 0.2	83.1 ± 0.1	83.8 ± 0.1	84.2 ± 0.2	84.0 ± 0.1	85.1 ± 0.1
Cars	**90.3 ± 0.2**	89.2 ± 0.1	**90.3 ± 0.1**	89.9 ± 0.0	89.4 ± 0.1	89.6 ± 0.1	87.8 ± 0.1	**90.5 ± 0.2**
DTD	**76.2 ± 0.3**	75.7 ± 0.5	75.4 ± 0.1	75.2 ± 0.4	74.5 ± 0.2	74.2 ± 0.2	73.7 ± 0.6	75.4 ± 0.3
Flowers	95.0 ± 0.1	94.7 ± 0.2	95.9 ± 0.2	93.9 ± 0.2	95.7 ± 0.2	**96.1 ± 0.1**	93.7 ± 0.2	95.5 ± 0.2
Food-101	**87.3 ± 0.1**	86.9 ± 0.1	**87.4 ± 0.1**	86.9 ± 0.1	86.9 ± 0.1	**87.2 ± 0.1**	85.9 ± 0.1	**87.4 ± 0.1**
Pets	**93.4 ± 0.1**	92.5 ± 0.0	**93.4 ± 0.1**	92.9 ± 0.1	92.2 ± 0.1	92.4 ± 0.1	92.1 ± 0.1	**93.3 ± 0.3**
SUN397	**64.8 ± 0.0**	64.0 ± 0.0	63.1 ± 0.1	61.7 ± 0.2	62.2 ± 0.2	62.0 ± 0.3	62.6 ± 0.1	62.8 ± 0.1
				90% Sparsity				
Aircraft	83.6 ± 0.4	82.8 ± 1.0	**83.9 ± 0.7**	**84.9 ± 0.3**	81.6 ± 0.5	83.0 ± 0.4	78.7 ± 0.4	**84.5 ± 0.4**
Birds	**72.4 ± 0.3**	68.5 ± 0.1	70.5 ± 0.1	67.8 ± 0.2	70.3 ± 0.0	**72.9 ± 0.2**	66.0 ± 0.2	71.6 ± 0.2
CIFAR-10	**97.4 ± 0.0**	96.6 ± 0.1	97.1 ± 0.0	96.6 ± 0.2	96.4 ± 0.1	97.0 ± 0.1	96.1 ± 0.1	97.0 ± 0.1
CIFAR-100	**85.6 ± 0.2**	83.9 ± 0.1	84.4 ± 0.0	83.0 ± 0.1	83.0 ± 0.2	83.7 ± 0.3	82.9 ± 0.2	84.4 ± 0.2
Caltech-101	**93.5 ± 0.1**	92.6 ± 0.2	92.9 ± 0.2	84.5 ± 0.3	91.7 ± 0.3	92.3 ± 0.4	90.9 ± 0.3	**93.9 ± 0.3**
Caltech-256	**86.1 ± 0.1**	84.8 ± 0.1	83.7 ± 0.3	78.6 ± 0.1	82.7 ± 0.2	84.0 ± 0.1	83.1 ± 0.2	84.0 ± 0.1
Cars	**90.3 ± 0.2**	88.5 ± 0.2	89.5 ± 0.0	89.5 ± 0.1	88.4 ± 0.1	89.2 ± 0.1	86.7 ± 0.2	**90.0 ± 0.2**
DTD	**76.2 ± 0.3**	75.2 ± 0.1	74.2 ± 0.1	71.9 ± 0.1	73.4 ± 0.4	75.2 ± 0.8	73.2 ± 0.4	75.5 ± 0.4
Flowers	95.0 ± 0.1	94.6 ± 0.1	95.3 ± 0.1	89.8 ± 0.2	95.5 ± 0.1	**96.1 ± 0.1**	93.4 ± 0.4	95.5 ± 0.3
Food-101	**87.3 ± 0.1**	86.6 ± 0.1	86.8 ± 0.1	86.4 ± 0.1	85.9 ± 0.1	**87.3 ± 0.2**	84.8 ± 0.0	**87.0 ± 0.1**
Pets	**93.4 ± 0.1**	92.1 ± 0.1	92.2 ± 0.1	91.1 ± 0.2	91.4 ± 0.2	92.3 ± 0.1	91.7 ± 0.2	92.7 ± 0.3
SUN397	**64.8 ± 0.0**	63.0 ± 0.0	62.5 ± 0.2	58.3 ± 0.2	61.3 ± 0.1	62.0 ± 0.2	61.2 ± 0.0	62.3 ± 0.1
				95% Sparsity				
Aircraft	**83.6 ± 0.4**	81.2 ± 0.4	N /A	82.6 ± 0.8	80.7 ± 0.1	82.5 ± 0.4	76.7 ± 0.8	**83.6 ± 0.6**
Birds	**72.4 ± 0.3**	66.9 ± 0.1	N /A	62.2 ± 0.1	68.3 ± 0.2	71.6 ± 0.1	62.3 ± 0.1	69.9 ± 0.1
CIFAR-10	**97.4 ± 0.0**	96.2 ± 0.1	N /A	95.5 ± 0.1	96.0 ± 0.1	96.6 ± 0.1	95.4 ± 0.1	96.7 ± 0.1
CIFAR-100	**85.6 ± 0.2**	82.9 ± 0.1	N /A	80.0 ± 0.1	82.0 ± 0.2	82.8 ± 0.0	80.9 ± 0.3	83.1 ± 0.1
Caltech-101	**93.5 ± 0.1**	91.9 ± 0.2	N /A	65.3 ± 0.8	90.7 ± 0.4	92.2 ± 0.3	89.8 ± 0.1	92.0 ± 0.3
Caltech-256	**86.1 ± 0.1**	83.1 ± 0.0	N /A	71.8 ± 0.3	81.1 ± 0.2	83.1 ± 0.2	80.3 ± 0.0	82.4 ± 0.1
Cars	**90.3 ± 0.2**	87.6 ± 0.1	N /A	87.5 ± 0.4	87.9 ± 0.3	88.9 ± 0.2	84.9 ± 0.2	88.9 ± 0.2
DTD	**76.2 ± 0.3**	74.1 ± 0.4	N /A	67.1 ± 0.8	73.3 ± 0.2	73.5 ± 0.2	72.6 ± 0.4	73.7 ± 0.3
Flowers	95.0 ± 0.1	94.1 ± 0.3	N /A	76.0 ± 1.3	94.9 ± 0.3	**96.0 ± 0.0**	93.0 ± 0.3	95.0 ± 0.3
Food-101	**87.3 ± 0.1**	85.5 ± 0.0	N /A	85.4 ± 0.1	85.1 ± 0.2	86.6 ± 0.0	83.0 ± 0.1	86.3 ± 0.1
Pets	**93.4 ± 0.1**	91.0 ± 0.1	N /A	84.5 ± 0.5	90.1 ± 0.2	91.6 ± 0.3	89.9 ± 0.3	92.3 ± 0.3
SUN397	**64.8 ± 0.0**	61.4 ± 0.2	N /A	51.4 ± 0.3	60.0 ± 0.3	61.1 ± 0.2	59.0 ± 0.1	60.9 ± 0.1
				98% Sparsity				
Aircraft	**83.6 ± 0.4**	79.1 ± 0.2	N /A	N /A	N /A	N /A	72.0 ± 0.2	81.4 ± 0.3
Birds	**72.4 ± 0.3**	63.4 ± 0.1	N /A	N /A	N /A	N /A	54.1 ± 0.1	65.4 ± 0.3
CIFAR-10	**97.4 ± 0.0**	95.0 ± 0.1	N /A	N /A	N /A	N /A	93.8 ± 0.1	96.0 ± 0.0
CIFAR-100	**85.6 ± 0.2**	79.8 ± 0.1	N /A	N /A	N /A	N /A	75.9 ± 0.2	80.7 ± 0.2
Caltech-101	**93.5 ± 0.1**	88.9 ± 0.1	N /A	N /A	N /A	N /A	85.2 ± 0.6	89.8 ± 0.3
Caltech-256	**86.1 ± 0.1**	80.3 ± 0.1	N /A	N /A	N /A	N /A	74.2 ± 0.0	78.9 ± 0.1
Cars	**90.3 ± 0.2**	85.5 ± 0.2	N /A	N /A	N /A	N /A	79.9 ± 0.5	86.8 ± 0.1
DTD	**76.2 ± 0.3**	72.6 ± 0.1	N /A	N /A	N /A	N /A	69.4 ± 0.3	71.8 ± 0.1
Flowers	**95.0 ± 0.1**	92.9 ± 0.1	N /A	N /A	N /A	N /A	91.8 ± 0.3	94.0 ± 0.2
Food-101	**87.3 ± 0.1**	83.2 ± 0.0	N /A	N /A	N /A	N /A	77.9 ± 0.1	84.2 ± 0.1
Pets	**93.4 ± 0.1**	88.8 ± 0.2	N /A	N /A	N /A	N /A	85.5 ± 0.1	89.8 ± 0.1
SUN397	**64.8 ± 0.0**	58.4 ± 0.1	N /A	N /A	N /A	N /A	53.8 ± 0.2	58.5 ± 0.1

Table B.3: (ResNet50) Full results showing transfer accuracy for sparse ResNet50 models with *full finetuning*.

B.2.3 Results for Extended Training Schedule

In this section we provide the full results for our experiments with extended training time for regularization methods (RigL and AC/DC), as described in Section 4.4.5. The transfer accuracies for linear finetuning are presented in Table B.4, while the same results for full finetuning are shown in Table B.5. Our experiments suggest that for regularization-based methods, the extra investment in upstream training can result in upstream models that transfer very well under both full and linear finetuning.

	Dense		AC/DC			RigL ERK	
	1x	2x	1x	3x	5x	1x	5x
				80% Sparsity			
Aircraft	49.2 ± 0.1	50.0 ± 0.1	**55.1 ± 0.1**	52.5 ± 0.2	N /A	54.6 ± 0.1	**55.2 ± 0.2**
Birds	57.7 ± 0.1	57.1 ± 0.0	58.4 ± 0.0	**59.2 ± 0.0**	N /A	55.2 ± 0.0	56.7 ± 0.1
CIFAR-10	91.2 ± 0.0	90.2 ± 0.0	90.9 ± 0.0	**91.4 ± 0.0**	N /A	89.7 ± 0.1	90.0 ± 0.1
CIFAR-100	74.6 ± 0.1	73.7 ± 0.0	74.7 ± 0.1	**75.3 ± 0.0**	N /A	73.1 ± 0.1	73.7 ± 0.0
Caltech-101	91.9 ± 0.1	**92.1 ± 0.2**	**92.4 ± 0.2**	**92.3 ± 0.2**	N /A	91.1 ± 0.1	90.8 ± 0.3
Caltech-256	84.8 ± 0.1	84.6 ± 0.1	84.6 ± 0.1	**85.4 ± 0.1**	N /A	83.3 ± 0.1	84.6 ± 0.1
Cars	53.4 ± 0.1	51.4 ± 0.1	56.6 ± 0.0	56.2 ± 0.1	N /A	57.4 ± 0.1	**58.6 ± 0.1**
DTD	73.5 ± 0.2	73.1 ± 0.2	**74.4 ± 0.1**	**74.1 ± 0.2**	N /A	73.5 ± 0.2	72.9 ± 0.3
Flowers	91.6 ± 0.1	91.1 ± 0.1	**92.7 ± 0.1**	**92.6 ± 0.1**	N /A	92.2 ± 0.1	92.3 ± 0.1
Food-101	73.2 ± 0.0	72.2 ± 0.0	73.8 ± 0.0	**74.8 ± 0.0**	N /A	73.3 ± 0.0	72.5 ± 0.1
Pets	**92.6 ± 0.1**	91.9 ± 0.1	92.3 ± 0.1	**92.8 ± 0.1**	N /A	91.9 ± 0.1	**92.5 ± 0.2**
SUN397	60.1 ± 0.0	59.9 ± 0.0	60.4 ± 0.0	**60.5 ± 0.0**	N /A	59.1 ± 0.1	59.9 ± 0.0
				90% Sparsity			
Aircraft	49.2 ± 0.1	50.0 ± 0.1	55.5 ± 0.1	54.6 ± 0.1	55.5 ± 0.0	54.1 ± 0.1	**56.6 ± 0.1**
Birds	57.7 ± 0.1	57.1 ± 0.0	58.7 ± 0.0	59.7 ± 0.1	**60.4 ± 0.1**	53.3 ± 0.0	57.2 ± 0.1
CIFAR-10	**91.2 ± 0.0**	90.2 ± 0.0	91.0 ± 0.0	90.9 ± 0.0	90.3 ± 0.0	90.0 ± 0.1	90.2 ± 0.1
CIFAR-100	**74.6 ± 0.1**	73.7 ± 0.0	74.3 ± 0.0	**74.7 ± 0.0**	74.2 ± 0.0	72.8 ± 0.1	73.4 ± 0.1
Caltech-101	91.9 ± 0.1	92.1 ± 0.2	92.5 ± 0.1	**92.9 ± 0.2**	**92.8 ± 0.2**	90.6 ± 0.3	91.4 ± 0.4
Caltech-256	84.8 ± 0.1	84.6 ± 0.1	84.5 ± 0.0	**85.3 ± 0.1**	**85.3 ± 0.1**	81.9 ± 0.0	84.5 ± 0.1
Cars	53.4 ± 0.1	51.4 ± 0.1	56.0 ± 0.1	58.7 ± 0.1	58.4 ± 0.1	55.5 ± 0.1	**60.5 ± 0.1**
DTD	73.5 ± 0.2	73.1 ± 0.2	73.7 ± 0.2	**74.2 ± 0.3**	**73.9 ± 0.2**	72.6 ± 0.3	72.7 ± 0.2
Flowers	91.6 ± 0.1	91.1 ± 0.1	92.4 ± 0.0	92.6 ± 0.0	**92.8 ± 0.0**	91.6 ± 0.1	92.4 ± 0.1
Food-101	73.2 ± 0.0	72.2 ± 0.0	73.8 ± 0.0	75.1 ± 0.0	**75.2 ± 0.0**	71.7 ± 0.0	72.6 ± 0.0
Pets	92.6 ± 0.1	91.9 ± 0.1	91.9 ± 0.1	**92.5 ± 0.1**	**92.6 ± 0.2**	91.1 ± 0.1	91.9 ± 0.2
SUN397	60.1 ± 0.0	59.9 ± 0.0	59.8 ± 0.1	60.3 ± 0.0	**61.2 ± 0.0**	57.7 ± 0.0	59.8 ± 0.1
				95% Sparsity			
Aircraft	49.2 ± 0.1	50.0 ± 0.1	56.6 ± 0.1	55.6 ± 0.0	N /A	53.5 ± 0.1	**56.9 ± 0.1**
Birds	57.7 ± 0.1	57.1 ± 0.0	57.7 ± 0.0	**59.2 ± 0.1**	N /A	51.9 ± 0.1	55.9 ± 0.0
CIFAR-10	**91.2 ± 0.0**	90.2 ± 0.0	90.5 ± 0.0	90.2 ± 0.0	N /A	89.4 ± 0.0	89.8 ± 0.1
CIFAR-100	**74.6 ± 0.1**	73.7 ± 0.0	73.4 ± 0.0	74.3 ± 0.1	N /A	71.5 ± 0.1	72.4 ± 0.1
Caltech-101	**91.9 ± 0.1**	**92.1 ± 0.2**	91.6 ± 0.1	**92.3 ± 0.3**	N /A	89.0 ± 0.1	91.4 ± 0.1
Caltech-256	**84.8 ± 0.1**	**84.6 ± 0.1**	82.8 ± 0.1	84.0 ± 0.1	N /A	80.1 ± 0.1	83.5 ± 0.1
Cars	53.4 ± 0.1	51.4 ± 0.1	56.9 ± 0.1	**57.4 ± 0.0**	N /A	52.9 ± 0.0	57.0 ± 0.1
DTD	73.5 ± 0.2	73.1 ± 0.2	72.7 ± 0.1	**74.7 ± 0.2**	N /A	71.9 ± 0.1	72.9 ± 0.2
Flowers	91.6 ± 0.1	91.1 ± 0.1	**93.0 ± 0.1**	92.5 ± 0.1	N /A	91.0 ± 0.1	92.4 ± 0.1
Food-101	73.2 ± 0.0	72.2 ± 0.0	73.2 ± 0.0	**74.7 ± 0.0**	N /A	70.6 ± 0.1	71.9 ± 0.0
Pets	**92.6 ± 0.1**	91.9 ± 0.1	91.0 ± 0.2	91.5 ± 0.1	N /A	90.1 ± 0.1	91.1 ± 0.1
SUN397	**60.1 ± 0.0**	59.9 ± 0.0	58.2 ± 0.0	59.7 ± 0.0	N /A	55.9 ± 0.1	58.3 ± 0.1

Table B.4: (ResNet50) Transfer accuracy for *extended training time* for ResNet50 with *linear finetuning*.

	Dense		AC/DC			RigL ERK	
	1x	2x	1x	3x	5x	1x	5x
			80% Sparsity				
Aircraft	83.6 ± 0.4	**84.3 ± 0.2**	83.3 ± 0.1	83.2 ± 0.3	N /A	82.6 ± 0.3	82.4 ± 0.2
Birds	72.4 ± 0.3	**73.5 ± 0.1**	69.9 ± 0.2	72.5 ± 0.2	N /A	72.3 ± 0.3	**73.4 ± 0.1**
CIFAR-10	**97.4 ± 0.0**	**97.4 ± 0.0**	96.9 ± 0.1	**97.5 ± 0.1**	N /A	96.9 ± 0.0	97.1 ± 0.0
CIFAR-100	**85.6 ± 0.2**	**85.8 ± 0.2**	84.9 ± 0.2	85.3 ± 0.1	N /A	83.6 ± 0.2	84.1 ± 0.4
Caltech-101	93.5 ± 0.1	**93.9 ± 0.1**	92.5 ± 0.2	93.4 ± 0.2	N /A	92.5 ± 0.1	92.0 ± 0.3
Caltech-256	86.1 ± 0.1	**86.5 ± 0.2**	85.4 ± 0.2	**86.7 ± 0.1**	N /A	83.8 ± 0.1	84.2 ± 0.2
Cars	**90.3 ± 0.2**	**90.5 ± 0.2**	89.2 ± 0.1	89.8 ± 0.3	N /A	89.4 ± 0.1	89.6 ± 0.1
DTD	76.2 ± 0.3	**76.9 ± 0.3**	75.7 ± 0.5	76.6 ± 0.0	N /A	74.5 ± 0.2	74.2 ± 0.2
Flowers	95.0 ± 0.1	95.5 ± 0.2	94.7 ± 0.2	95.1 ± 0.2	N /A	95.7 ± 0.2	**96.1 ± 0.1**
Food-101	87.3 ± 0.1	**87.5 ± 0.1**	86.9 ± 0.1	**87.7 ± 0.1**	N /A	86.9 ± 0.1	**87.2 ± 0.1**
Pets	**93.4 ± 0.1**	**93.4 ± 0.2**	92.5 ± 0.0	**93.4 ± 0.2**	N /A	92.2 ± 0.1	92.4 ± 0.1
SUN397	64.8 ± 0.0	**65.1 ± 0.0**	64.0 ± 0.0	64.8 ± 0.1	N /A	62.2 ± 0.2	62.0 ± 0.3
			90% Sparsity				
Aircraft	83.6 ± 0.4	**84.3 ± 0.2**	82.8 ± 1.0	82.6 ± 0.2	**83.5 ± 0.4**	81.6 ± 0.5	83.0 ± 0.4
Birds	72.4 ± 0.3	**73.5 ± 0.1**	68.5 ± 0.1	71.6 ± 0.2	72.8 ± 0.2	70.3 ± 0.0	72.9 ± 0.2
CIFAR-10	**97.4 ± 0.0**	**97.4 ± 0.0**	96.6 ± 0.1	97.0 ± 0.1	97.1 ± 0.1	96.4 ± 0.1	97.0 ± 0.1
CIFAR-100	**85.6 ± 0.2**	**85.8 ± 0.2**	83.9 ± 0.1	84.7 ± 0.1	85.3 ± 0.1	83.0 ± 0.2	83.7 ± 0.3
Caltech-101	93.5 ± 0.1	**93.9 ± 0.1**	92.6 ± 0.2	93.0 ± 0.0	93.2 ± 0.1	91.7 ± 0.3	92.3 ± 0.4
Caltech-256	86.1 ± 0.1	**86.5 ± 0.2**	84.8 ± 0.1	86.1 ± 0.1	**86.5 ± 0.1**	82.7 ± 0.2	84.0 ± 0.1
Cars	**90.3 ± 0.2**	**90.5 ± 0.2**	88.5 ± 0.2	89.3 ± 0.1	89.8 ± 0.1	88.4 ± 0.1	89.2 ± 0.1
DTD	76.2 ± 0.3	**76.9 ± 0.3**	75.2 ± 0.1	75.3 ± 0.2	76.4 ± 0.2	73.4 ± 0.4	75.2 ± 0.8
Flowers	95.0 ± 0.1	95.5 ± 0.2	94.6 ± 0.1	95.4 ± 0.1	**95.9 ± 0.1**	95.5 ± 0.1	**96.1 ± 0.1**
Food-101	87.3 ± 0.1	**87.5 ± 0.1**	86.6 ± 0.1	87.4 ± 0.1	**87.7 ± 0.1**	85.9 ± 0.1	87.3 ± 0.2
Pets	**93.4 ± 0.1**	**93.4 ± 0.2**	92.1 ± 0.1	92.6 ± 0.1	93.0 ± 0.1	91.4 ± 0.2	92.3 ± 0.1
SUN397	64.8 ± 0.0	**65.1 ± 0.0**	63.0 ± 0.0	64.3 ± 0.0	64.8 ± 0.1	61.3 ± 0.1	62.0 ± 0.2
			95% Sparsity				
Aircraft	83.6 ± 0.4	**84.3 ± 0.2**	81.2 ± 0.4	82.2 ± 0.3	N /A	80.7 ± 0.1	82.5 ± 0.4
Birds	72.4 ± 0.3	**73.5 ± 0.1**	66.9 ± 0.1	70.1 ± 0.1	N /A	68.3 ± 0.2	71.6 ± 0.1
CIFAR-10	**97.4 ± 0.0**	**97.4 ± 0.0**	96.2 ± 0.1	96.6 ± 0.1	N /A	96.0 ± 0.1	96.6 ± 0.1
CIFAR-100	**85.6 ± 0.2**	**85.8 ± 0.2**	82.9 ± 0.1	84.0 ± 0.1	N /A	82.0 ± 0.2	82.8 ± 0.0
Caltech-101	93.5 ± 0.1	**93.9 ± 0.1**	91.9 ± 0.2	92.9 ± 0.1	N /A	90.7 ± 0.4	92.2 ± 0.3
Caltech-256	86.1 ± 0.1	**86.5 ± 0.2**	83.1 ± 0.0	85.3 ± 0.0	N /A	81.1 ± 0.2	83.1 ± 0.2
Cars	**90.3 ± 0.2**	**90.5 ± 0.2**	87.6 ± 0.1	89.0 ± 0.1	N /A	87.9 ± 0.3	88.9 ± 0.2
DTD	76.2 ± 0.3	**76.9 ± 0.3**	74.1 ± 0.4	75.0 ± 0.4	N /A	73.3 ± 0.2	73.5 ± 0.2
Flowers	95.0 ± 0.1	95.5 ± 0.2	94.1 ± 0.3	95.0 ± 0.2	N /A	94.9 ± 0.3	**96.0 ± 0.0**
Food-101	**87.3 ± 0.1**	**87.5 ± 0.1**	85.5 ± 0.0	86.7 ± 0.1	N /A	85.1 ± 0.2	86.6 ± 0.0
Pets	**93.4 ± 0.1**	**93.4 ± 0.2**	91.0 ± 0.1	91.6 ± 0.1	N /A	90.1 ± 0.2	91.6 ± 0.3
SUN397	64.8 ± 0.0	**65.1 ± 0.0**	61.4 ± 0.2	63.0 ± 0.0	N /A	60.0 ± 0.3	61.1 ± 0.2

Table B.5: (ResNet50) Transfer accuracy for *extended training time* for ResNet50 with *full finetuning*.

B.3 Finetuning Experiments on ResNet18 and ResNet34

In this section, we further validate our findings for linear finetuning from ResNet50 on two additional smaller architectures, namely ResNet18 and ResNet34. Specifically, we test whether regularization pruning methods generally have better transfer potential than progressive sparsification methods, and whether regularization pruning methods improve over dense models for fine-grained classification tasks. For this purpose, we trained AC/DC and GMP on ImageNet using ResNet18 and ResNet34 models, for 80% and 90% sparsity, using the same hyperparameters as for ResNet50. For both ResNet18 and ResNet34, there was a fairly large gap in ImageNet validation accuracy between GMP and AC/DC for both 80% and 90% sparsity, in favor of GMP, which almost recovered the baseline accuracy at 80% sparsity.

We show the results for linear finetuning using AC/DC and GMP for ResNet18 and ResNet34 in Table B.6. Interestingly, despite the larger gap in ImageNet validation accuracy between GMP and AC/DC (with GMP being closer to the dense baseline), AC/DC tends to outperform GMP in terms of transfer performance, on most of the downstream tasks. Furthermore, we observe that AC/DC tends to transfer better than the dense baseline, especially for specialized or fine-grained downstream tasks. These observations confirm our findings for linear finetuning on ResNet50.

Pruning Strategy Task	Dense	GMP 80%	GMP 90%	AC/DC 80%	AC/DC 90%
ResNet18					
Aircraft	47.7 ± 0.1	45.5 ± 0.1	45.6 ± 0.1	48.0 ± 0.1	**48.1 ± 0.1**
Birds	49.4 ± 0.1	49.3 ± 0.1	48.1 ± 0.0	**50.2 ± 0.0**	48.7 ± 0.1
CIFAR-10	87.2 ± 0.0	**87.4 ± 0.0**	87.2 ± 0.0	**87.4 ± 0.0**	87.2 ± 0.1
CIFAR-100	68.9 ± 0.0	68.1 ± 0.0	69.1 ± 0.0	**69.6 ± 0.1**	68.9 ± 0.0
Caltech-101	89.4 ± 0.3	**89.8 ± 0.3**	88.6 ± 0.2	89.0 ± 0.2	88.2 ± 0.4
Caltech-256	**79.4 ± 0.1**	78.3 ± 0.1	77.3 ± 0.1	78.8 ± 0.1	77.3 ± 0.1
Cars	45.6 ± 0.1	45.0 ± 0.1	44.4 ± 0.1	46.2 ± 0.1	**46.7 ± 0.1**
DTD	68.1 ± 0.1	68.2 ± 0.3	66.9 ± 0.2	**68.6 ± 0.2**	68.4 ± 0.2
Flowers	89.0 ± 0.1	89.3 ± 0.1	89.3 ± 0.1	89.9 ± 0.1	**90.2 ± 0.1**
Food-101	64.9 ± 0.0	65.0 ± 0.0	64.6 ± 0.0	**65.6 ± 0.0**	65.3 ± 0.0
Pets	**90.1 ± 0.1**	89.8 ± 0.1	89.4 ± 0.2	89.7 ± 0.1	89.4 ± 0.1
SUN397	**54.8 ± 0.1**	53.8 ± 0.1	52.9 ± 0.1	**54.8 ± 0.1**	53.5 ± 0.1
ResNet34					
Aircraft	45.8 ± 0.2	43.5 ± 0.2	44.9 ± 0.1	48.7 ± 0.1	**50.7 ± 0.2**
Birds	52.9 ± 0.0	53.0 ± 0.1	53.0 ± 0.1	**54.5 ± 0.1**	54.2 ± 0.1
CIFAR-10	89.5 ± 0.0	89.1 ± 0.0	88.5 ± 0.0	**89.6 ± 0.0**	89.0 ± 0.0
CIFAR-100	71.0 ± 0.0	70.4 ± 0.1	70.2 ± 0.1	**72.0 ± 0.0**	**72.0 ± 0.0**
Caltech-101	**92.5 ± 0.2**	91.8 ± 0.3	90.9 ± 0.2	92.0 ± 0.3	91.8 ± 0.4
Caltech-256	82.2 ± 0.1	81.8 ± 0.0	81.4 ± 0.1	**82.3 ± 0.1**	81.2 ± 0.1
Cars	47.3 ± 0.1	46.0 ± 0.1	45.6 ± 0.1	48.5 ± 0.1	**49.0 ± 0.1**
DTD	69.5 ± 0.1	68.6 ± 0.5	68.6 ± 0.2	**70.4 ± 0.3**	69.6 ± 0.2
Flowers	88.1 ± 0.1	88.5 ± 0.1	89.0 ± 0.1	90.0 ± 0.1	**91.1 ± 0.1**
Food-101	66.8 ± 0.0	66.7 ± 0.0	67.4 ± 0.0	68.2 ± 0.0	**68.8 ± 0.0**
Pets	92.0 ± 0.1	**92.5 ± 0.1**	91.4 ± 0.1	91.7 ± 0.1	91.1 ± 0.2
SUN397	55.9 ± 0.1	55.4 ± 0.1	55.0 ± 0.1	**56.8 ± 0.1**	55.6 ± 0.1

Table B.6: (ResNet18/ResNet34) Transfer accuracy for sparse models with *linear finetuning*.

B.4 Finetuning Experiments on MobileNetV1

The MobileNet [HZC+17] architecture is a natural choice for devices with limited computational resources. We measure the results of sparse transfer with full and linear finetuning on the same downstream tasks starting from dense ImageNet models pruned using regularization-based and progressive sparsification methods. Specifically, we use AC/DC, STR for regularization methods and M-FAC [FKA21] for the progressive sparsification category.

M-FAC is a framework for efficiently computing high-dimensional inverse-Hessian vector products, which can be applied to different scenarios that use second-order information. In particular, one such instance is pruning, where M-FAC aims to solve the same optimization

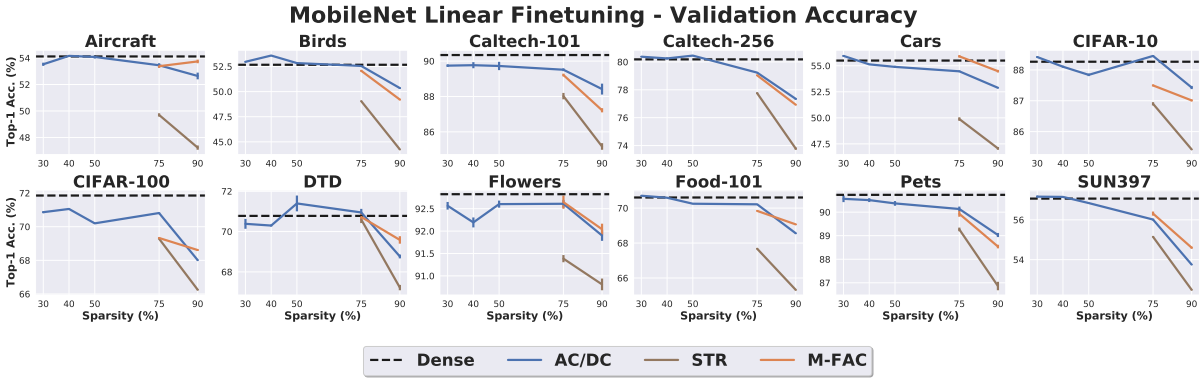


Figure B.3: (MobileNetV1) Per-dataset downstream validation accuracy for transfer learining with *linear finetuning*.

problem as WoodFisher, and thus from this point of view these methods are very similar. In particular, it has been shown [FKA21] that M-FAC outperforms WoodFisher on ImageNet models, in terms of accuracy at a given sparsity level. Specifically, for MobileNet, M-FAC surpasses all existing methods at 90% sparsity, reaching 67.2% validation accuracy. For this reason, we included M-FAC, in favor of WoodFisher, to our list of progressive sparsification methods for MobileNetV1.

Due to the smaller size of the MobileNetV1 architecture, we additionally test the effect that lower sparsity levels have on the transfer performance, by training on ImageNet AC/DC models at 30%, 40% and 50% sparsity; these models fully recover the dense baseline accuracy on ImageNet.

B.4.1 Linear Finetuning

Model Task	Dense	AC/DC 30%	AC/DC 40%	AC/DC 50%	AC/DC 75%	AC/DC 90%	M-FAC 75%	M-FAC 89%	STR 75%	STR 90%
Aircraft	54.1 ± 0.2	53.5 ± 0.1	**54.2 ± 0.1**	54.1 ± 0.1	53.5 ± 0.2	52.6 ± 0.3	53.4 ± 0.1	53.7 ± 0.1	49.7 ± 0.1	47.2 ± 0.2
Birds	52.7 ± 0.1	53.0 ± 0.1	**53.6 ± 0.1**	52.8 ± 0.0	52.6 ± 0.1	50.3 ± 0.1	52.1 ± 0.1	49.2 ± 0.1	49.0 ± 0.0	44.2 ± 0.0
CIFAR-10	88.3 ± 0.1	88.4 ± 0.0	88.1 ± 0.0	87.8 ± 0.0	**88.5 ± 0.0**	87.4 ± 0.1	87.5 ± 0.0	87.0 ± 0.0	86.9 ± 0.1	85.4 ± 0.0
CIFAR-100	**71.9 ± 0.0**	70.9 ± 0.1	71.1 ± 0.0	70.2 ± 0.0	70.8 ± 0.0	68.0 ± 0.0	69.3 ± 0.0	68.6 ± 0.0	69.3 ± 0.0	66.3 ± 0.0
Caltech-101	**90.3 ± 0.1**	89.7 ± 0.1	89.8 ± 0.2	89.7 ± 0.2	89.5 ± 0.1	88.4 ± 0.3	89.2 ± 0.1	87.2 ± 0.1	88.0 ± 0.2	85.2 ± 0.2
Caltech-256	80.2 ± 0.1	80.4 ± 0.0	80.2 ± 0.1	**80.5 ± 0.0**	79.2 ± 0.0	77.3 ± 0.1	79.0 ± 0.0	76.9 ± 0.0	77.8 ± 0.1	73.8 ± 0.1
Cars	55.5 ± 0.0	**55.9 ± 0.1**	55.1 ± 0.1	54.9 ± 0.1	54.5 ± 0.1	52.9 ± 0.0	**55.9 ± 0.1**	54.5 ± 0.1	49.9 ± 0.2	47.1 ± 0.1
DTD	70.8 ± 0.2	70.4 ± 0.2	70.3 ± 0.0	**71.4 ± 0.4**	70.9 ± 0.2	68.8 ± 0.1	70.7 ± 0.2	69.6 ± 0.2	70.6 ± 0.2	67.2 ± 0.1
Flowers	**92.8 ± 0.1**	92.6 ± 0.1	92.2 ± 0.1	92.6 ± 0.1	92.6 ± 0.1	91.9 ± 0.1	92.6 ± 0.1	92.0 ± 0.1	91.4 ± 0.1	90.8 ± 0.1
Food-101	70.6 ± 0.0	**70.7 ± 0.0**	70.6 ± 0.0	70.3 ± 0.0	70.2 ± 0.0	68.6 ± 0.0	69.8 ± 0.0	69.1 ± 0.0	67.7 ± 0.0	65.3 ± 0.0
Pets	**90.7 ± 0.1**	90.6 ± 0.1	90.5 ± 0.1	90.4 ± 0.1	90.1 ± 0.1	89.0 ± 0.1	89.9 ± 0.1	88.5 ± 0.1	89.3 ± 0.1	86.9 ± 0.2
SUN397	57.1 ± 0.0	**57.2 ± 0.1**	57.2 ± 0.0	56.8 ± 0.0	56.0 ± 0.0	53.8 ± 0.0	56.3 ± 0.1	54.6 ± 0.0	55.1 ± 0.0	52.5 ± 0.0

Table B.7: (MobileNet) Transfer accuracy for *linear finetuning* using sparse MobileNet models

Our results for linear finetuning are presented in Figure B.3 and Table B.7; we note that each experiment was run from five different random seeds, and we report the mean and standard deviation. We observe that AC/DC and M-FAC outperform STR at both 75% and 90% sparsity. Furthermore, we do not observe the same trend where regularization methods outperformed progressive sparsification. For example, AC/DC tends to be close to or outperform M-FAC at 75% sparsity, while at 90% sparsity M-FAC performs better on almost half of the tasks. Differently from ResNet50, for MobileNet neither regularization based nor progressive sparsification models outperform the dense baseline, at higher sparsity (75% and 90%). We observe at lower sparsity (30% and 50%) a few instances where sparse models

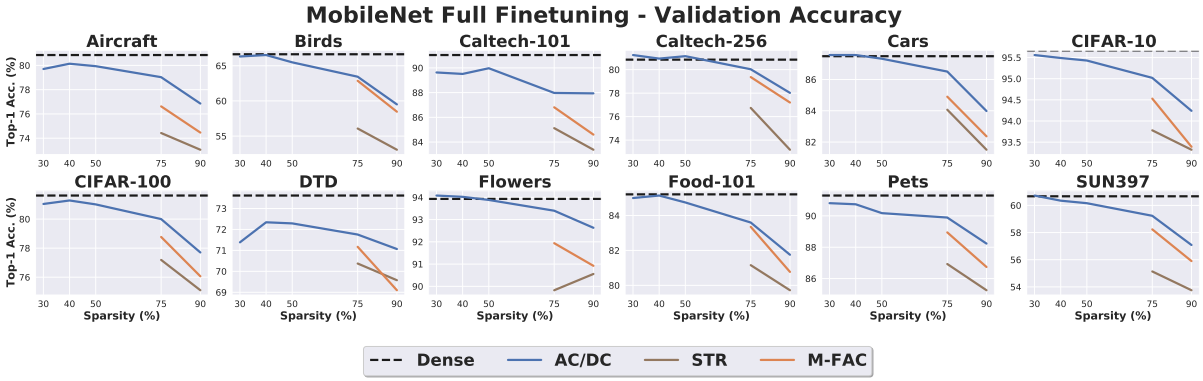


Figure B.4: (MobileNetV1) Per-dataset downstream validation accuracy for transfer learning with *full finetuning*.

slightly outperform the dense baseline (Birds, Cars, DTD), but generally the differences are not significant.

B.4.2 Full Finetuning

The results for full finetuning using MobileNet models are presented in Figure B.4 and Table B.8; we note that in this case each experiment was run once. We observe that the performance of sparse models decays more quickly than for ResNet50, and even at lower sparsity (30-50%) there is a gap in transfer performance compared to the dense baseline. Furthermore, AC/DC outperforms STR and M-FAC at both 75% and 90% sparsity on all downstream tasks. Overall, the results for MobileNet indicate that the transfer performance is significantly affected by the sparsity of the backbone model, for both linear and full finetuning. Moreover, the experiments on MobileNet seem to suggest that although some of the conclusions derived from the ResNet experiments are confirmed (e.g. sparse models usually have similar or slightly better performance to the dense baseline for linear finetuning), the guidelines for the preferred sparsity method in a given scenario might be specific to the choice of the backbone architecture.

In conclusion, for both linear and full finetuning, we observe that generally the performance decays faster with increased sparsity, compared to ResNet50; this is expected, given the lower parameter count for MobileNet and the larger gap in ImageNet validation accuracy between dense and sparse models.

Pruning Strategy	Dense	AC/DC 30%	AC/DC 40%	AC/DC 50%	AC/DC 75%	AC/DC 90%	M-FAC 75%	M-FAC 89%	STR 75%	STR 90%
Aircraft	**80.9**	79.7	80.1	79.9	79.0	76.9	76.6	74.5	74.4	73.0
Birds	**66.6**	66.3	66.5	65.5	63.4	59.5	62.9	58.5	56.1	53.1
CIFAR-10	**95.7**	95.6	95.5	95.4	95.0	94.2	94.5	93.4	93.8	93.3
CIFAR-100	**81.6**	81.0	81.3	81.0	80.0	77.7	78.8	76.1	77.2	75.1
Caltech-101	**91.0**	89.6	89.5	90.0	88.0	87.9	86.8	84.6	85.1	83.4
Caltech-256	80.9	**81.2**	80.9	81.1	80.0	78.0	79.4	77.2	76.7	73.2
Cars	87.5	**87.6**	**87.6**	87.3	86.5	84.0	84.9	82.4	84.1	81.5
DTD	**73.6**	71.4	72.3	72.3	71.8	71.1	71.2	69.1	70.4	69.6
Flowers	93.9	**94.1**	94.0	93.9	93.4	92.6	91.9	90.9	89.8	90.6
Food-101	**85.2**	85.0	85.1	84.7	83.6	81.8	83.4	80.8	81.2	79.7
Pets	**91.3**	90.8	90.7	90.2	89.9	88.2	88.9	86.7	86.9	85.3
SUN397	**60.7**	**60.7**	60.3	60.2	59.2	57.1	58.2	55.9	55.1	53.8

Table B.8: (MobileNet) Transfer accuracy for *full finetuning* using sparse MobileNet models

B.4.3 Accuracy Trade-Off MobileNet / Sparse ResNet50

Finally, we consider the accuracy trade-off of using a smaller network such as MobileNet (4.2M trainable weights) versus a larger model, ResNet50 (25.5M trainable weights), but pruned to 80% or 90% sparsity. We present linear and full finetuning accuracy results for these two scenarios for an easier comparison in Tables B.9 and B.10. We use the overall best pruning strategy for each type of transfer on ResNet50: AC/DC for linear finetuning and WoodFisher for full finetuning. Note that these same results are also presented in Tables B.3, B.2 for ResNet50 and Tables B.8, B.7 for MobileNet.

We observe that generally, pruning ResNet50 to 80% or even 90% sparsity results in higher accuracy than MobileNet, for both linear and full finetuning. However, in almost all cases, the gap is below 5%. This finding confirms conventional wisdom that training and pruning large networks generally results in higher accuracy than training dense small networks from scratch.

Model	MobileNet Dense	ResNet50–AC/DC 80%	ResNet50–AC/DC 90%
Aircraft	54.1 ± 0.2	55.1 ± 0.1	55.5 ± 0.1
Birds	52.7 ± 0.1	58.4 ± 0.0	58.7 ± 0.0
CIFAR-10	88.3 ± 0.1	90.9 ± 0.0	91.0 ± 0.0
CIFAR-100	71.9 ± 0.0	74.7 ± 0.1	74.3 ± 0.0
Caltech-101	90.3 ± 0.1	92.4 ± 0.2	92.5 ± 0.1
Caltech-256	80.2 ± 0.1	84.6 ± 0.1	84.5 ± 0.0
Cars	55.5 ± 0.0	56.6 ± 0.0	56.0 ± 0.1
DTD	70.8 ± 0.2	74.4 ± 0.1	73.7± 0.2
Flowers	92.8 ± 0.1	92.7 ± 0.1	92.4 ± 0.0
Food-101	70.6 ± 0.0	73.8 ± 0.0	73.8 ± 0.0
Pets	90.7 ± 0.1	92.3 ± 0.1	91.9 ± 0.1
SUN397	57.1 ± 0.0	60.4 ± 0.0	59.8 ± 0.1

Table B.9: Comparison of MobileNet dense vs. ResNet50 sparse models when transferring with *linear finetuning*

Model	MobileNet Dense	ResNet50–WoodFisher 80%	ResNet50–WoodFisher 90%
Aircraft	80.9	84.8 ± 0.2	84.5 ± 0.4
Birds	66.6	72.4 ± 0.4	71.6 ± 0.2
CIFAR-10	95.7	97.2 ± 0.1	97.0 ± 0.1
CIFAR-100	81.6	85.1 ± 0.1	84.4 ± 0.2
Caltech-101	91.0	93.7 ± 0.1	93.9 ± 0.3
Caltech-256	80.9	85.1 ± 0.1	84.0± 0.1
Cars	87.5	90.5 ± 0.2	90.0 ± 0.2
DTD	73.6	75.4 ± 0.3	75.5 ± 0.4
Flowers	93.9	95.5 ± 0.2	95.5 ± 0.3
Food-101	85.2	87.4 ± 0.1	87.0 ± 0.1
Pets	91.3	93.3 ± 0.3	92.7 ± 0.3
SUN397	60.7	62.8 ± 0.1	62.3 ± 0.1

Table B.10: Comparison of MobileNet dense vs. ResNet50 sparse models when transferring with *full finetuning*

B.5 Finetuning Experiments Using Structured Sparsity

In this section, we examine the transfer properties of models that were sparsified using structured pruning methods, which remove entire convolutional filters. Specifically, we use both ResNet50 and MobileNetV1 models trained on ImageNet and we do full finetuning on all twelve downstream tasks.

B.5.1 ResNet50 with Structured Sparsity

We consider a ResNet50 model that was pruned with progressive sparsification, using the L_1 magnitude of the convolutional filters as a pruning criterion. The resulting model has an ImageNet validation accuracy of 75.7% and results in 2.2x inference speed-up compared to the dense baseline, when evaluated on a single sample; this makes it comparable to unstructured 90% sparse models that achieve a similar inference speed-up (please see Table 4.5). The results for full finetuning with the structured sparse model, together with the best results for dense and unstructured 80% and 90% models are presented in Table B.11. We observe that models with structured sparsity transfer similarly to or worse than unstructured 90% sparse models. Note that the unstructured ResNet50 model has higher ImageNet accuracy

compared to 90% sparse models, at a similar inference speed-up. These results align with the observations made in Section 4.4.4, that having fewer filters in the structured sparse models limits their capability of expressing features.

Dataset	Dense	Structured	Unstructured	
			80%	90%
Aircraft	83.6 ± 0.4	81.8 ± 0.5	**84.8± 0.2**	**84.9 ± 0.3**
Birds	72.4 ± 0.3	70.7 ± 0.1	**73.4± 0.1**	72.9 ± 0.2
Caltech101	**93.5± 0.1**	92.8 ± 0.1	**93.7± 0.1**	**93.9 ± 0.3**
Caltech256	**86.1 ± 0.1**	84.6 ± 0.1	85.4 ± 0.2	84.8 ± 0.1
Cars	90.3 ± 0.2	89.4 ± 0.0	**90.5 ± 0.2**	90.0 ± 0.2
CIFAR-10	**97.4 ± 0.**	97.1 ± 0.1	97.2 ± 0.1	$97.1 \pm 0.$
CIFAR-100	**85.6 ± 0.2**	84.7 ± 0.2	85.1 ± 0.1	84.4 ± 0.2
DTD	**76.2 ± 0.3**	75.2 ± 0.2	**75.7 ± 0.5**	75.5 ± 0.4
Flowers	95.0 ± 0.1	95.2 ± 0.0	**96.1 ± 0.1**	**96.1 ± 0.1**
Food-101	**87.3 ± 0.1**	86.3 ± 0.1	**87.4 ± 0.1**	**87.3 ± 0.2**
Pets	**93.4 ± 0.1**	92.5 ± 0.1	**93.4 ± 0.2**	92.7 ± 0.3
SUN397	**64.8 ± 0.**	63.4 ± 0.1	$64.0 \pm 0.$	$63.0 \pm 0.$

Table B.11: (ResNet50) Comparison on full finetuning between dense baseline, models with structured sparsity, and best results for unstructured 80% and 90% sparsity.

Dataset	Dense	50% Time	50% FLOPs
Aircraft	80.9	**82.9**	**83.0**
Birds	**66.6**	66.1	66.1
Caltech101	**91.0**	88.6	88.9
Caltech256	**80.9**	78.6	78.4
Cars	87.5	**88.4**	**88.3**
CIFAR-10	**95.7**	95.2	95.3
CIFAR-100	**81.6**	79.9	80.2
DTD	**73.6**	71.1	72.2
Flowers	93.9	**94.1**	**94.1**
Food-101	**85.2**	84.6	84.5
Pets	**91.3**	91.0	91.0
SUN397	**60.7**	59.4	59.1

Table B.12: (MobileNet) Full finetuning validation accuracy for MobileNet models with structured sparsity, at 50% inference time or 50% inference FLOPs.

B.5.2 MobileNet with Structured Sparsity

We additionally perform full finetuning using MobileNet models pruned for structured sparsity. For these experiments, we use the upstream models provided in [HLL+18]; specifically, we use the MobileNet models that achieve 50% of the inference time or have 50% of the dense FLOPs. These models achieve 70.2% and 70.5% ImageNet validation accuracy, respectively. The results presented in Table B.12 show that in general models with structured sparsity perform similar to or worse than their dense counterparts, with the exception of Aircraft and Cars where these models significantly outperform the dense baseline.

B.6 Sparse Transfer Learning for Segmentation

To complement the experiments for object detection, we executed transfer learning for a YOLACT model [BZXL19] using a ResNet-101 backbone, that has been trained and sparsified on the segmentation version of the COCO dataset. The average sparsity of the model is $\sim$ 87%, obtained via gradual magnitude pruning (GMP). The model has mAP@0.5 values 49.36 (bounding box), and 46.37 (mask), versus 50.16 (bounding box), 46.57 (mask) for the dense model on COCO. We transfer the pruned trained weights onto the Pascal dataset. The prediction heads get initialized as dense, and kept dense for transfer. The results are presented in Tables B.13 and B.14, and show that indeed sparse transfer is competitive against the dense variant in this case as well.

B.7 Additional Factors Influencing Sparse Transfer

In this section, we further provide ablation studies meant to investigate the impact of different model or training hyperparameter choices on transfer performance. First, we investigate whether the use of label smoothing when training the upstream model can negatively impact the sparse transfer accuracy with linear finetuning; this phenomenon has been previously

Type	all	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
box	32.62	54.05	51.96	48.72	44.81	40.84	34.72	26.89	17.33	6.33	0.55
mask	30.74	50.28	47.66	44.57	41.02	36.39	31.47	25.53	18.55	10.03	1.91

Table B.13: Mean average precision for dense transfer on Pascal, at various thresholds.

Type	all	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
box	33.55	54.15	51.79	49.2	45.57	41.51	35.95	29.2	19.66	7.78	0.74
mask	31.5	50.66	47.89	45.04	41.67	37.32	32.39	26.35	19.98	11.22	2.5

Table B.14: Mean average precision for *sparse* transfer on Pascal, at various thresholds. Notice the similar or slightly improved accuracy.

documented for dense models in [KSL19]. Second, we investigate the impact of using bias in the fully connected layer when doing full finetuning; this is motivated by the fact that some ResNet50 versions we considered (e.g. the one used in STR [KRS+20]) does not originally use bias.

B.7.1 Impact of label smoothing on transfer accuracy

We take advantage of the fact that we have STR checkpoints trained with and without label smoothing (LS) to investigate the effect of LS on dense and sparse transfer accuracy in the context of linear transfer. As Table B.15 shows, label smoothing tends to have a negative effect on transfer accuracy (confirming the results in [KSL19]). However, our experiments suggest that this effect is more pronounced on the Aircraft and Cars datasets in the case of sparse STR models, and generally for most specialized datasets for the dense models. Furthermore, we observe that the performance gap tends to narrow with increased sparsity. We also note that even with label smoothing, at 80% sparsity STR matches or outperforms GMP on all datasets, although the effect largely reverses at 90% sparsity.

Overall, these data can be taken as a preliminary confirmation of the importance of controlling for variation in hyperparameters when comparing the transfer performance of various training and pruning methods.

Dataset	Dense	Dense LS	STR 80%	STR LS 80%	STR 90%	STR LS 90%	STR 95%	STR LS 95%	STR 98%	STR LS 98%
Aircraft	49.2 ± 0.1	38.2 ± 0.1	53.7 ± 0.0	47.0 ± 0.0	52.9 ± 0.1	46.4 ± 0.1	50.3 ± 0.1	46.6 ± 0.1	48.0 ± 0.1	45.2 ± 0.1
Birds	57.7 ± 0.1	52.4 ± 0.0	56.2 ± 0.1	56.4 ± 0.0	55.2 ± 0.1	56.0 ± 0.0	52.1 ± 0.1	51.7 ± 0.1	43.7 ± 0.0	45.6 ± 0.0
CIFAR-10	91.2 ± 0.0	89.6 ± 0.0	91.4 ± 0.0	90.1 ± 0.0	90.6 ± 0.0	89.4 ± 0.0	89.1 ± 0.0	88.6 ± 0.0	86.5 ± 0.0	86.0 ± 0.0
CIFAR-100	74.6 ± 0.1	71.6 ± 0.0	74.7 ± 0.0	73.3 ± 0.0	73.7 ± 0.1	72.2 ± 0.1	71.7 ± 0.0	70.1 ± 0.0	67.4 ± 0.0	66.3 ± 0.0
Caltech-101	91.9 ± 0.1	91.6 ± 0.1	91.2 ± 0.1	92.6 ± 0.1	90.9 ± 0.1	91.1 ± 0.2	90.0 ± 0.2	89.8 ± 0.1	86.3 ± 0.1	85.4 ± 0.1
Caltech-256	84.8 ± 0.1	84.6 ± 0.1	83.6 ± 0.0	84.3 ± 0.0	82.6 ± 0.0	82.6 ± 0.1	80.2 ± 0.1	79.7 ± 0.0	73.4 ± 0.1	73.8 ± 0.0
Cars	53.4 ± 0.1	44.9 ± 0.1	57.0 ± 0.1	50.9 ± 0.0	54.8 ± 0.1	49.8 ± 0.1	50.5 ± 0.1	46.9 ± 0.1	44.4 ± 0.1	42.5 ± 0.1
DTD	73.5 ± 0.2	72.3 ± 0.1	74.3 ± 0.2	73.9 ± 0.3	73.8 ± 0.1	73.7 ± 0.2	72.1 ± 0.2	71.9 ± 0.1	68.4 ± 0.2	68.3 ± 0.1
Flowers	91.6 ± 0.1	86.7 ± 0.1	93.0 ± 0.0	91.2 ± 0.0	93.0 ± 0.1	92.1 ± 0.1	91.9 ± 0.1	91.0 ± 0.1	90.8 ± 0.1	90.4 ± 0.1
Food-101	73.2 ± 0.0	69.5 ± 0.0	73.9 ± 0.0	72.2 ± 0.0	72.6 ± 0.0	71.1 ± 0.0	70.7 ± 0.0	68.8 ± 0.0	65.3 ± 0.0	64.3 ± 0.0
Pets	92.6 ± 0.1	92.9 ± 0.1	91.7 ± 0.0	92.4 ± 0.1	91.1 ± 0.1	91.7 ± 0.1	89.8 ± 0.1	90.1 ± 0.1	85.5 ± 0.1	86.6 ± 0.1
SUN397	60.1 ± 0.0	59.3 ± 0.1	60.3 ± 0.0	60.0 ± 0.1	58.2 ± 0.0	58.5 ± 0.1	56.3 ± 0.0	55.8 ± 0.0	50.9 ± 0.0	51.0 ± 0.0

Table B.15: (ResNet50) Linear Finetuning Validation Accuracy of STR-pruned and dense models with and without label smoothing.

B.7.2 Impact of fully connected layer bias on full finetuning transfer accuracy

Dataset	With FC Bias	Without FC Bias
Aircraft	79.8 ± 0.6	79.8 ± 0.3
Birds	67.9 ± 0.2	68.1 ± 0.1
CIFAR-10	96.5 ± 0	96.5 ± 0.1
CIFAR-100	83.7 ± 0.2	83.6 ± 0.2
Caltech-101	91.2 ± 0.2	90.7 ± 0.6
Caltech-256	84.4 ± 0.1	84.0 ± 0.1
Cars	87.7 ± 0.1	87.8 ± 0.1
DTD	74.4 ± 0.2	73.7 ± 0.6
Flowers	94 ± 0.1	93.7 ± 0.2
Food-101	86 ± 0.1	85.9 ± 0.1
Pets	92.1 ± 0.1	92.1 ± 0.1
SUN397	63.2 ± 0.1	62.6 ± 0.1

Table B.16: (ResNet50) Top-1 validation transfer accuracy for STR, with using bias in the FC layer vs. without. The original model architecture does not use bias in the FC layer.

Dataset	With FC Bias	Without FC Bias
Aircraft	74.2	74.4
Birds	56.4	56.1
CIFAR-10	93.9	93.8
CIFAR-100	77.5	77.2
Caltech-101	86.3	85.1
Caltech-256	76.9	76.7
Cars	83.8	84.1
DTD	71.8	70.4
Flowers	90.4	89.8
Food-101	80.9	81.2
Pets	87.9	86.9
SUN397	56.1	55.1

Table B.17: (MobileNetV1) Top-1 validation transfer accuracy for STR, with using bias in the FC layer vs. without. The original model architecture does not use bias in the FC layer.

In our experiments, we used the original architectures used to train the upstream ImageNet models when performing transfer with full-finetuning, only resizing the final layer to match the number of output classes in the downstream task. This choice was necessitated partially by ensuring that the weights were applied correctly. For example, the RigL models were trained using TensorFlow, which uses slightly different Convolution and MaxPooling padding conventions than PyTorch. Likewise, STR models were trained using a slightly nonstandard PyTorch implementation of ResNet50, which did not use a bias term in the final Fully-Connected (FC) layer. We investigate the possibility that the latter difference could have an effect on downstream transfer accuracy. To do so, we transferred a set of 80% sparse ResNet50 STR models to all downstream tasks, using a bias term in the FC layer. The results are shown in Table B.16. Additionally, we perform a similar comparison on MobileNetV1, for STR models at 75% sparsity. As in the case of ResNet50, the version of MobileNet used by the STR models does not use bias in the final classification layer. The results illustrating the bias effect on full finetuning for MobileNet are presented in Table B.17. We observe that the presence of a bias term in the final layer can, in some cases, have a small positive effect on the resulting model, and so we caution that these effects be considered when choosing a transfer architecture.

APPENDIX C

Appendix for Chapter 5

C.1 Theoretical Support for the CrAM Update

In this section, we attempt to formally derive a generic training method whose purpose is to provide compressible models, which perform well even after being compressed. To understand why we can hope to achieve such guarantees, we first take a brief detour to the area of robust optimization.

C.1.1 Robust Optimization

Generally, practical training methods are based on versions of stochastic gradient descent attempting to minimize a loss function $L(\boldsymbol{\theta})$. However, $\boldsymbol{\theta}$ might turn out to be a bad solution as the landscape of L in its neighborhood could contain large changes in value. To address this issue, one may attempt to flatten L such that it is less sensitive to sharp drops in value localized around a very small region. To this extent, a standard robustification can be defined by

$$\widetilde{L}(\boldsymbol{\theta}) = \max_{\|\boldsymbol{\delta}\| \leq \rho} L(\boldsymbol{\theta} + \boldsymbol{\delta}) \,, \tag{C.1}$$

which makes the value of $\widetilde{L}(\boldsymbol{\theta})$ take that of the largest value of L given by perturbation of $\boldsymbol{\theta}$ within a ball of radius ρ. While this robustified function may seem well suited to generic training tasks, it is a priori unclear that it is amenable to optimization.

However, under certain conditions, we can efficiently optimize $\widetilde{L}$ by using a classical theorem in robust optimization due to Danskin [Dan12].

Theorem C.1.1. *(Danskin) Let $\mathcal{C} \subseteq \mathbb{R}^m$ be a compact set, let a function $\phi : \mathbb{R}^n \times \mathcal{C} \to \mathbb{R}$ such that $\phi(\cdot, \boldsymbol{y})$ is continuously differentiable for every fixed $\boldsymbol{y} \in \mathcal{C}$ and $\nabla_{\boldsymbol{x}} \phi(\boldsymbol{x}, \boldsymbol{y})$ is continuous on $\mathbb{R}^n \times \mathcal{C}$, and let $\psi : \mathbb{R}^n \to \mathbb{R}$ be defined as*

$$\psi(x) = \max_{\boldsymbol{y} \in \mathcal{C}} \phi(\boldsymbol{x}, \boldsymbol{y}) \ .$$

Then ψ is locally Lipschitz continuous, directionally differentiable, and its directional derivatives satisfy

$$d\psi(\boldsymbol{x}; \boldsymbol{d}) = \max_{\boldsymbol{y} \in \mathcal{C}^*} \boldsymbol{d}^\top \nabla_{\boldsymbol{x}} \phi(\boldsymbol{x}, \boldsymbol{y}) \ .$$

where $\mathcal{C}^*(\boldsymbol{x})$ is the set of maximizers

$$\mathcal{C}^*(\boldsymbol{x}) = \left\{ \boldsymbol{y}^* : \phi(\boldsymbol{x}, \boldsymbol{y}^*) = \max_{\boldsymbol{y} \in \mathcal{C}} \phi(\boldsymbol{x}, \boldsymbol{y}) \right\} .$$

In particular, if for some $\boldsymbol{x} \in \mathbb{R}^n$ the set $\mathcal{C}^*(\boldsymbol{x}) = \{\boldsymbol{y}_{\boldsymbol{x}}^*\}$ is a singleton, then ψ is differentiable at $\boldsymbol{x}$ and

$$\nabla \psi(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \phi(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{x}}^*) .$$

This shows that, under certain assumptions, we can obtain directional derivatives for $\widetilde{L}(\boldsymbol{\theta})$ by simply maximizing $L(\boldsymbol{\theta} + \boldsymbol{\delta})$ over $\boldsymbol{\delta} \in B_2(\rho)$.

Corollary C.1.1. *Let $\widetilde{L}$ be defined as in Equation (C.1), and define*

$$\mathcal{C}^*(\boldsymbol{\theta}) = \left\{ \boldsymbol{\delta} : \|\boldsymbol{\delta}\| \leq \rho, L(\boldsymbol{\theta} + \boldsymbol{\delta}) = \max_{\|\boldsymbol{\delta}^*\| \leq \rho} L(\boldsymbol{\theta} + \boldsymbol{\delta}^*) \right\} ,$$

and let $\overline{\boldsymbol{\delta}} \in \mathcal{C}^*(\boldsymbol{\theta})$. Provided that $L(\boldsymbol{\theta})$ is continuously differentiable, and $\boldsymbol{\theta}$ is not an articulation point for $\widetilde{L}$, $-\nabla L(\boldsymbol{\theta} + \overline{\boldsymbol{\delta}})$ is a descent direction for $\widetilde{L}(\boldsymbol{\theta})$ as long as it is nonzero.

Proof. Let $\boldsymbol{h} = \nabla L(\boldsymbol{\theta} + \overline{\boldsymbol{\delta}})$. We apply Danskin's theorem for $\phi(\boldsymbol{\theta}, \boldsymbol{\delta}) = L(\boldsymbol{\theta} + \boldsymbol{\delta})$ and $\mathcal{C} = B_2(\rho)$. This shows that

$$d\widetilde{L}(\boldsymbol{\theta}; \boldsymbol{h}) = \sup_{\boldsymbol{\delta} \in \mathcal{C}^*(\boldsymbol{\theta})} \boldsymbol{h}^\top \nabla L(\boldsymbol{\theta} + \boldsymbol{\delta}) \geq \boldsymbol{h}^\top \nabla L(\boldsymbol{\theta} + \overline{\boldsymbol{\delta}}) = \boldsymbol{h}^\top \boldsymbol{h} \geq 0 .$$

Provided that $\boldsymbol{\theta}$ is not an articulation point for $\widetilde{L}$, we also have that $d\widetilde{L}(\boldsymbol{\theta}; -\boldsymbol{h}) = -d\widetilde{L}(\boldsymbol{\theta}; \boldsymbol{h}) \leq 0$, which concludes the proof. □

From Robust Optimization to SAM

Per Corollary C.1.1, to obtain a descent direction it suffices to maximize $L(\boldsymbol{\theta} + \boldsymbol{\delta})$ over the set of perturbations satifying $\|\boldsymbol{\delta}\| \leq \rho$. In general, even when the underlying function L is convex, this may be a difficult problem. Instead, one may simply attempt to obtain a good local maximizer of L in a bounded region around $\boldsymbol{\theta}$. The simplest possible way to do so is by performing a step of *gradient ascent*, which can be regarded as a proxy for the maximization subproblem. Using this step, we immediately obtain the iteration:

$$\widetilde{\boldsymbol{\theta}}_t = \boldsymbol{\theta}_t + \frac{\rho}{\|\nabla L(\boldsymbol{\theta}_t)\|} \nabla L(\boldsymbol{\theta}_t) , \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\widetilde{\boldsymbol{\theta}}_t) , \tag{C.2}$$

which recovers the extrapolated SAM gradient step from [FKMN21].

There is exhaustive research that has previously been done on robust optimization methods. For a comprehensive reference, we point the reader to Teo's PhD thesis [Teo07].

C.1.2 Robust Optimization for Compressible Models

With the robust optimization framework in mind, we are ready to attempt implementing a similar scheme which exhibits robustness to compression.

To motivate the method, let us consider the post-training compression. After training the model to weights to $\boldsymbol{\theta}_T$ we apply a one-shot compression method C over some perturbation

$\boldsymbol{\theta}_T + \boldsymbol{\delta}$ of the weights. This captures several iterative methods for compression, such as iterative pruning, where changes in weights are alternated with one-shot pruning methods.

If our goal is to make the loss after compression robust within a small neighborhood of perturbations $\boldsymbol{\delta}$, we can establish as a formal objective to minimize the robustified loss

$$L^{\mathsf{CrAM}}(\boldsymbol{\theta}) := \max_{\boldsymbol{\delta}:\|\boldsymbol{\delta}\|\leq\rho} L\left(C\left(\boldsymbol{\theta}+\boldsymbol{\delta}\right)\right), \tag{C.3}$$

for some magnitude ρ of allowed perturbations. In our case we will focus on the case where these are bounded in ℓ_2 norm, but this can be easily extended to other choices of the domain. Just as before, we can now attempt to minimize L^{CrAM}, or find a near-stationary point, by gradient descent. Using the robust optimization framework we may attempt to optimize it using Corollary C.1.1 after replacing $L(\cdot)$ with $L(C(\cdot))$.

Naturally, this poses some obstacles in our case. The main one is the fact that it is not true that $L(C(\boldsymbol{\theta}))$ will generally be continuously differentiable, so the conditions required to obtain descent directions via an inner maximization loop are not satisfied. However, we can show that under certain conditions, continuous differentiability fails only at a set of points of measure 0.

Definition C.1.1. *Let S be a countable set, let $\{P_i\}_{i\in S}$ be a covering of $\mathbb{R}^n$ with convex sets, and let $S(\boldsymbol{x})$ denote the family of indices from S for which $\boldsymbol{x}\in P_i$. Let a family of projection operators $\{\Pi_i\}_{i\in S}$, such that for any $\boldsymbol{x}$ the projections $\{\Pi_i(\boldsymbol{x})\}_{i\in S(\boldsymbol{x})}$ all map to the same point. We call a projective compression operator with respect to $\{\Pi_i\}_{i\in S}$ a mapping $C:\mathbb{R}^n\to\mathbb{R}^n$ such that*

$$C(\boldsymbol{x}) = \Pi_i(\boldsymbol{x}), \quad \text{for any } i\in S(\boldsymbol{x}).$$

For example, in the case of the *Top-K* compression operator, we can define a covering of $\mathbb{R}^n$ with sets P such that all elements $x\in P$ share the indices $A\subseteq[n]$, $|A|=k$ for the Top-K elements in absolute value (with ties broken lexicographically), and furthermore, all elements from P preserve the signs across A. It is clear that any $x\in\mathbb{R}^n$ belongs in some such set P, and since there are a finite number of subsets of size k and of possible signs for the components in $[n]$, we have a finite covering. Assume, without loss of generality, that a set P from the covering consists of elements for which the first k components are the highest in absolute value, and the signs across these components are shared across all $x\in P$. Then, for any $\boldsymbol{x},\boldsymbol{y}\in P$, $\lambda\in(0,1)$ and $i\leq k$, we have that $|\lambda\boldsymbol{x}_i+(1-\lambda)\boldsymbol{y}_i| = \lambda|\boldsymbol{x}_i|+(1-\lambda)|\boldsymbol{y}_i|$ (since $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ share the same sign). Using that $\lambda|\boldsymbol{x}_i|+(1-\lambda)|\boldsymbol{y}_i|\geq\lambda|\boldsymbol{x}_j|+(1-\lambda)|\boldsymbol{y}_j|$, for any $k<j<n$, together with the triangle inequality, we obtain that $|\lambda\boldsymbol{x}_i+(1-\lambda)\boldsymbol{y}_i|\geq|\lambda\boldsymbol{x}_j+(1-\lambda)\boldsymbol{y}_j|$, for any $i\leq k$ and $j>k$. Therefore, any P satisfying the conditions described above is a convex set. We can further define a projection for each subset A of coordinates of cardinality k. Then, it is clear that for any given vector $\boldsymbol{x}$, the set $A\in S(\boldsymbol{x})$ iff the largest k coordinates of $\boldsymbol{x}$ in absolute value (with ties broken lexicographically) are supported in A. Therefore, we can conclude that Top-K is a projective compression operator.

Lemma C.1.1 (Continuously differentiable functions induce few singularities after compression)**.** *Let $L:\mathbb{R}^n\to\mathbb{R}$ be a continuously differentiable function, and let C be a projective compression operator. Then the function $g(\boldsymbol{x}) := L(C(\boldsymbol{x}))$ is continuously differentiable everywhere except at a set of points of measure 0. Furthermore, so is the robustified function $L^{\mathsf{CrAM}}(\boldsymbol{x}) := \max_{\|\boldsymbol{\delta}\|\leq\rho} L(C(\boldsymbol{x}+\boldsymbol{\delta}))$.*

Proof. First we note that the boundary of any convex set has measure zero by standard arguments in convex analysis [Lan86]. Since a countable union of sets of measure zero has measure zero, it follows that the union of the boundaries of P_i's has measure zero. Now since L is continuously differentiable, within any set P_i, we have that $g(\boldsymbol{x}) = L(\Pi_i(\boldsymbol{x}))$, and hence it remains continuously differentiable. Therefore the only region for which we can not argue about continuous differentiability is the complement of the union of interiors of P_i's, $(\cup_i \text{int} P_i)^c \subseteq \cup_i \partial P_i$ which is a set of measure zero. Since g is well-behaved almost everywhere, all that remains to argue is that this is the same case with L^{CrAM}.

For any fixed direction $\Delta\boldsymbol{\theta}$, we define the mapping

$$M(\boldsymbol{\theta}) = \Delta\boldsymbol{\theta}^\top \nabla g(\boldsymbol{\theta})\,,$$

and its robustification

$$\widetilde{M}(\boldsymbol{\theta}) = \max_{\|\boldsymbol{\delta}\| \leq \rho} M(\boldsymbol{\theta} + \boldsymbol{\delta}) = \max_{\|\boldsymbol{\delta}\| \leq \rho} \Delta\boldsymbol{\theta}^\top \nabla g(\boldsymbol{\theta} + \boldsymbol{\delta})\,.$$

Hence the directional derivative w.r.t. $\Delta\boldsymbol{\theta}$ of $L^{\text{CrAM}}(\boldsymbol{\theta})$ is discontinuous only when $\widetilde{M}(\boldsymbol{\theta})$ is discontinuous. Finally, we note that this almost never happens, as M is continuous almost everywhere, and thus so must be $\widetilde{M}$. Thus, all directional derivatives are continuous except at a set of measure 0, which concludes the proof. □

Finally, just like in the previous case, maximizing $L(C(\boldsymbol{\theta} + \boldsymbol{\delta}))$ over small perturbations is generally intractable. So we instead consider obtaining a good enough maximizer via a standard iterative method which has shown good performance in practice. More precisely we consider the projected gradient ascent method, which provides strong theoretical guarantees, even when the projection is performed onto non-convex domains [PIVA21]. In the case where the compression operator represents magnitude pruning, this corresponds to the iterative hard thresholding (IHT) method, frequently employed in the sparse recovery literature.

To reach a good iterate within this specific domain we instead perform a single step of (projected) gradient ascent, which matches the IHT iteration:

$$\widetilde{\boldsymbol{\theta}}_t = C\left(\boldsymbol{\theta}_t + \rho \cdot \nabla L(\boldsymbol{\theta}_t)\right)\,. \tag{C.4}$$

We have therefore obtained a re-derivation of the CrAM update in Equation 5.2.

Furthermore, we note that the analysis above holds also for the CrAM$^+$. Namely, we can rewrite the CrAM$^+$ loss as $L^{\text{CrAM}^+} = \max_{\|\boldsymbol{\delta}\| \leq \rho}(L(C(\boldsymbol{\theta} + \boldsymbol{\delta})) + L(\boldsymbol{\theta}))$, and apply Lemma C.1.1 to obtain that L^{CrAM^+} is continuously differentiable almost everywhere, and so are its directional derivatives.

C.2 Additional Image Classification Experiments

C.2.1 Variability of Batch Norm Tuning Results

We emphasize that CrAM relies on a small calibration set of training samples to correct the Batch Norm statistics, namely running mean and variance, after pruning, particularly at high sparsity. We call this procedure Batch Norm Tuning (BNT). To ensure that the accuracies we report for sparse models are stable under the choice of the calibration set, we perform

10 independent trials of BNT, on 10 randomly chosen subsets of 1000 training samples, for each model and for different sparsity levels. The results of this experiment are presented in Table C.1, which also contains the "raw" numbers used in Figure 5.1. Notice that the accuracy after one-shot pruning and BNT is very stable, with respect to the choice of the calibration set. In particular, for the CrAM$^+$ model, the standard deviation is $\leq 0.1\%$ across all sparsity levels considered. We also report the "raw" one-shot pruning accuracy (i.e. before BNT) for CrAM models in Table C.2.

Model	Dense	Sparsity				
		50%	60%	70%	80%	90%
Baseline	77.22	75.87 ± 0.09	73.82 ± 0.07	68.86 ± 0.08	51.96 ± 0.27	8.57 ± 0.11
SAM	77.35	76.47 ± 0.04	75.11 ± 0.1	71.87 ± 0.07	60.20 ± 0.13	18.25 ± 0.18
CrAM-k50	77.48	77.3 ± 0.07	76.61 ± 0.05	74.77 ± 0.08	68.23 ± 0.11	33.04 ± 0.16
CrAM$^+$-k70	77.32	77.22 ± 0.05	77.1 ± 0.05	77.15 ± 0.05	76.3 ± 0.08	61.92 ± 0.11
CrAM$^+$-Multi	77.28	77.24 ± 0.06	77.05 ± 0.05	77.0 ± 0.04	75.8 ± 0.07	74.74 ± 0.04

Table C.1: (ImageNet/ResNet50) Validation accuracy for the dense models, and after one-shot pruning using global magnitude pruning, followed by BNT on 1000 samples. The results for one-shot pruning are the mean accuracies, and their standard deviations, when BNT is performed on 10 different random calibration sets, of 1000 training samples each.

Model	Dense	Sparsity				
		50%	60%	70%	80%	90%
Baseline	77.22	74.35	68.9	46.36	2.0	0.1
SAM	77.35	75.02	70.4	52.8	3.66	0.11
CrAM-k50	77.48	75.91	73.54	63.05	13.59	0.16
CrAM$^+$-k70	77.32	77.03	76.6	76.3	72.6	3.6
CrAM$^+$-Multi	77.28	76.23	74.87	75.69	72.32	52.25

Table C.2: (ImageNet/ResNet50) Validation accuracy for the dense models, and after one-shot pruning using global magnitude, before BNT.

C.2.2 Comparison With Other Methods on CIFAR-10

In this section we provide additional results accompanying those presented in Section 5.4.3. Namely, we provide comparison between one-shot pruning CrAM$^+$-Multi vs. standard dense baselines (SGD, SAM), and we provide numbers before and after BNT for sparse models on VGG-16 and ResNet18.

Model	Dense	Sparsity				
		50%	60%	70%	80%	90%
Baseline	93.0 ± 0.1	92.2 ± 0.0	91.0 ± 0.3	88.0 ± 0.2	78.0 ± 1.1	45.8 ± 3.0
SAM	**93.5 ± 0.1**	92.8 ± 0.2	92.4 ± 0.0	90.7 ± 0.3	85.2 ± 0.4	54.6 ± 1.7
CrAM$^+$-Multi	93.2 ± 0.1	93.2 ± 0.1	93.1 ± 0.1	92.9 ± 0.1	92.4 ± 0.1	90.3 ± 0.1

Table C.3: (CIFAR-10/ResNet20) Test acc. (%) for the dense models, and after one-shot pruning (+BNT). The baseline is the model after SGD training. For all models we apply one-shot pruning at different sparsity (+BNT), but no additional retraining. Results are averaged across 3 runs from different seeds.

In Table C.3 we show the accuracy of the dense baseline, SAM and CrAM$^+$-Multi on ResNet20, before and after one-shot pruning at different sparsities; for CrAM$^+$-Multi we consider the version from Section 5.6.2, namely the sparsity values are samples uniformly at each step in the range $30 - 90\%$. The results after one-shot pruning are presented after BNT over a random subset of 1000 train samples, over 100 batches. We note there are small variations in the results after BNT, due to the choice of the random calibration set. These variations are small ($\pm 0.1/0.2\%$) for CrAM$^+$-Multi models, across all sparsity levels considered, but they are larger for the one-shot pruned dense baselines at high sparsity (e.g. 80% and 90%). Moreover, the accuracy before BNT is still high for CrAM at lower sparsity levels (e.g. 91.9% at 70% sparsity), but it degrades at high sparsity (e.g. 50.5% at 90% sparsity). We believe that this is only due to the BatchNorm statistics, which are adapted to the dense model during training, but they no longer reflect the distribution shift after weight pruning. This is confirmed by the fact that 90% sparse models improve to over 90% test accuracy after only a few iterations of BNT, and are very robust to the choice of the calibration set.

Architecture	BNT	Sparsity 50%	80%	90%	93%	95%	97%	98%
ResNet18	No	95.7±0.1	95.3±0.2	93.6±0.7	92.0±1.4	89.8±2.2	81.4±6.3	48.7±5.8
	Yes	95.6±0.0	95.7±0.0	95.5±0.1	95.5±0.1	95.5±0.1	95.2±0.0	94.5±0.3
VGG-16	No	94.2±0.1	93.9±0.2	86.7±1.6	48.5±1.7	19.8±15.4	16.0±10.2	12.4±4.1
	Yes	94.2±0.1	94.2±0.1	94.0±0.1	94.0±0.2	94.1± 0.1	93.8±0.2	93.0±0.2

Table C.4: (CIFAR-10) Test accuracy (%) for the sparse models obtained with one-shot-pruning from CrAM$^+$-k95, before and after BNT. Results are averaged across 3 runs from different seeds.

Moreover, we show the extended results of CrAM$^+$-k95 discussed in Section 5.4.3, before and after BNT, on ResNet18 and VGG-16. From Table C.4 we can see that one-shot pruning CrAM$^+$-k95 without BNT preserves accuracy up to 80% sparsity, after which BNT is required to correct the BatchNorm statistics. Remarkably, the VGG-16 models at 97% and 98% sparsity have very low accuracy, which is improved greatly by BNT. Furthermore, also in this highly sparse regimes the accuracy is very robust with respect to the choice of the calibration set for BNT.

C.3 Language Models - reproducibility and hyperparameters

Adam, SAM and CrAM optimization. We use the finetuning recipe for SQuADv1.1 with the established hyperparameters provided in [DCLT19, WDS$^+$20]; namely, we start from the pre-trained [1]`bert-base-uncased`, and use `batch-size=16`, `max-sequence-length=384`, `doc-stride=128`. For other hyper-parameters we conduct a grid search for each optimizer. The set of hyperparameters used for grid search are the following:

- `learning-rate` $\in \{3\text{e}{-5}, 5\text{e}{-5}, 8\text{e}{-5}\}$
- `num-train-epochs` $\in \{2, 3\}$ for SAM and CrAM, and `num-train-epochs` $\in \{2, 3, 4, 6\}$ for Adam

[1]Available for download at https://huggingface.co/bert-base-uncased)

- `label-smoothing-factor` $\in \{0.0, 0.1, 0.2\}$
- for CrAM/SAM: ρ in the range $1e{-4}$ to $1e{-1}$.

Following this search, we pick the set of hyperparameters that produces the best results after one-shot magnitude pruning to 50% sparsity. The best hyperparameters are as follows:

- Adam: `num-train-epochs=2`, `learning-rate=`$8e{-5}$, `label-smoothing-ratio=0.1`
- SAM: `num-train-epochs=2`, `learning-rate=`$8e{-5}$, `label-smoothing-ratio=0.0`, ρ=`0.01`
- CrAM (all CrAM runs use the same set of hyperparameters): `num-train-epochs=3`, `learning-rate=`$8e{-5}$, `label-smoothing-ratio=0.2`, ρ=`0.005`

One-shot pruning. We apply one-shot pruning with two different pruners: magnitude and oBERT. For one-shot magnitude pruning we impose uniform sparsity distribution over all layers. For one-shot oBERT pruning we adopt the suggested set of hyper-parameters by authors, which we briefly describe here for completeness: 1024 gradients, dampening $1e{-7}$, block-size 50, 4 recomputations, global sparsity distribution over all layers. For more details please refer to the oBERT paper [KCN+22].

Sparse fine-tuning of one-shot pruned models. We fine-tune one-shot oBERT-pruned models with the fixed sparsity mask and Adam optimizer. To identify the best set of hyperparameters for fine-tuning of the sparse model, we conduct a grid search over the following parameters: `learning-rate` $\in \{3e{-5}, 5e{-5}, 8e{-5}, 1e{-4}\}$, `num-train-epochs` $\in \{1, 2\}$, `label-smoothing-ratio` $\in \{0.0, 0.2\}$, `warmup-ratio` $\in \{0.0, 0.1\}$. We freeze the embedding layer and employ early-stopping technique to prevent overfitting.

Speed-ups of pruned BERT-base models. In Table 5.11 we present speed-ups of our pruned models in the sparsity-aware CPU inference engine DeepSparse [KKG+20, Dee21] (version 1.0.2). We consider two different scenarios and report speed-ups relative to the dense model benchmarked in the same environment.